DESIGN AND CHARACTERIZATION OF A THRUST VANE POSITION CONTROLLER FOR EXHAUST FLOW DEFLECTION TVC WITH DYNAMICALLY CHANGING LOADS

A Thesis
Presented to
The Academic Faculty

By

Jacob Grey Sobering

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

DESIGN AND CHARACTERIZATION OF A THRUST VANE POSITION CONTROLLER FOR EXHAUST FLOW DEFLECTION TVC WITH DYNAMICALLY CHANGING LOADS

Approved by:

Dr. David Taylor, Advisor
School of Electrical Engineering
Georgia Institute of Technology

Dr. Thomas Habetler
School of Electrical Engineering
Georgia Institute of Technology

Dr. Patricio Vela
School of Electrical Engineering
Georgia Institute of Technology

Date Approved: April 10, 2018

ACKNOWLEDGEMENTS

I would like to first thank my adviser, Dr. David Taylor, for his support and guidance on this research project. His extensive knowledge and patience are what truly made this thesis successful. Second, I would like to thank my parents, Tim Sobering and Jayne Bendure, for imparting on me the desire to pursue a career in engineering and teaching me the importance of seeing things through to the end. Finally, I would like to thank Adrienne Meyer for her unending support, encouragement, and love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AC	Alternating Current
ADC	Analog-to-Digital Converter
CG	Center of Gravity
CP	Center of Pressure
DC	Direct Current
GPIO	General Purpose Input/Output
HWIL	Hardware-in-the-Loop
IC	Integrated Circuit
ISR	Interrupt Service Routine
LED	Light Emitting Diode
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
SSI	Synchronous Serial Interface Protocol
TVC	Thrust Vector Control
$\bar{A}$	Equilibrium State Space A Matrix
A_{RM}	Area of the Rocket Nozzle
A_{V}	Surface Area of the Thrust Vane
A_{VE}	Velocity Estimation Filter Constant
a	Acceleration
$\bar{B}$	Equilibrium State Space B Matrix
b	ADC Number of Bits
b_{SYS}	Friction Co-efficient of the System
D_{CP}	Linear Distance Between Thrust Vane Rotational Axis and CP
D_{CG}	Linear Distance Between Thrust Vane Rotational Axis and CG
F_{CG}	Force Acting on the Center of Mass Due to Acceleration
F_{CP}	Force Acting on the Center of Pressure Due to Aerodynamics

F_{RM}	Peak Thrust of Rocket Motor
$G(z)$	Tick Integrator Z-Domain Transfer Function
G_{Amp}	Current Sense Amplifier Gain
G_{D}	Drive Motor Gear Ratio
G_{L}	Load Motor Gear Ratio
$H(z)$	Differentiating Filter Z-Domain Transfer Function
$H(s)_{\mathrm{Butterworth}}$	Butterworth Filter Transfer Function
I_{D}	Drive Motor Current
I_{L}	Load Motor Current
I_{Motor}	Motor Current
J_{SYS}	System Inertia
J_V	Thrust Vane Inertia
$K_{11\mathrm{DP}}$	Gain K_{11} of Drive Motor Position Control Loop
$K_{12\mathrm{DP}}$	Gain K_{12} of Drive Motor Position Control Loop
$K_{2\mathrm{DP}}$	Gain K_2 of Drive Motor Position Control Loop
$K_{1\mathrm{DT}}$	Gain K_1 of Drive Motor Torque Control Loop
$K_{2\mathrm{DT}}$	Gain K_2 of Drive Motor Torque Control Loop
$K_{1\mathrm{LT}}$	Gain K_1 of Load Motor Torque Control Loop
$K_{2\mathrm{LT}}$	Gain K_2 of Load Motor Torque Control Loop
k_{tD}	Drive Motor Torque Constant
k_{tL}	Load Motor Torque Constant
L_{D}	Drive Motor Inductance
L_{L}	Load Motor Inductance
m	Mass
m_{V}	Thrust Vane Mass
N_{ENC}	Binary Integer Value Read in From Encoder
N_{ADC}	Binary Integer ADC Value
p	Pressure
Q	Filter Quality Factor
R_{Sense}	Motor Current Sense Resistance
R_{D}	Drive Motor Resistance

R_{L}	Load Motor Resistance
$r_{\mathrm{DP}}(t)$	Drive Motor Position Control Loop Reference Command
$r_{\mathrm{DT}}(t)$	Drive Motor Torque Control Loop Reference Command
$r_{\mathrm{LT}}(t)$	Load Motor Torque Control Loop Reference Command
T_{CG}	Torque Due to Mass Properties
T_{CP}	Torque Due to Aerodynamic Forces
T_{J}	Torque Due to Inertia of Thrust Vane
T_{DS}	Torque Out of Drive Motor After Gear Head
T_{LS}	Torque Out of Load Motor After Gear Head
T_{D}	Torque Out of Drive Motor Before Gear Head
T_{L}	Torque Out of Load Motor Before Gear Head
T_{SYS}	Total Torque Acting on the System
T_{Tick}	Tick Integrator Sample Period in Seconds
T_{VE}	Differentiating Filter Sample Period
T_{sample}	Discrete Time Sample Period
T_{cont}	Continuous Time Sample Period
$\bar{u}$	Equilibrium State Space Input
$\tilde{u}$	Difference Between Estimated and Actual State Space Input
$u_{\mathrm{eD}}(t)$	Drive Motor Electrical State Space System Input
$u_{\mathrm{eL}}(t)$	Load Motor Electrical State Space System Input
$u_{\mathrm{m}}(t)$	Mechanical State Space System Input
V_{CS}	Voltage at Current Sense Amplifier Output
V_{D}	Drive Motor Terminal Voltage
V_{L}	Load Motor Terminal Voltage
V_{REF}	Current Sense Amplifier Reference Voltage
V_{Range}	ADC Voltage Sense Range
V_{Sense}	Voltage at Current Sense Amplifier Input
$w_{\mathrm{DT}}(t)$	Drive Motor Torque Control Loop Disturbance Input
$w_{\mathrm{LT}}(t)$	Load Motor Torque Control Loop Disturbance Input
$w_{\mathrm{eD}}(t)$	Drive Motor Electrical State Space Disturbance Input
$w_{\mathrm{eL}}(t)$	Load Motor Electrical State Space Disturbance Input

$\bar{x}$	Equilibrium State Space Variables
$\tilde{x}$	Difference Between Estimated and Actual State Space Variable
$x_{\mathrm{eD}}(t)$	Drive Motor Electrical State Space Variables
$x_{\mathrm{eL}}(t)$	Load Motor Electrical State Space Variables
$x_{\mathrm{m}}(t)$	Mechanical State Space Variables
$y_{\mathrm{eD}}(t)$	Drive Motor Electrical State Space System Output
$y_{\mathrm{eL}}(t)$	Load Motor Electrical State Space System Output
$y_{\mathrm{m}}(t)$	Mechanical State Space System Output
$\widehat{A}_{\mathrm{DP}}$	Drive Motor Position Control Loop Modified A Matrix
$\widehat{A}_{\mathrm{DT}}$	Drive Motor Torque Control Loop Modified A Matrix
Δ_{ADC}	ADC Quantization Step
θ_{D}	Drive Motor Encoder Position
θ_{L}	Load Motor Encoder Position
θ_{SYS}	Coupled Motor Shaft Position
θ_{Vane}	Thrust Vane Angular Position with Respect to Vertical
ω_{D}	Drive Motor Angular Velocity
ω_{L}	Load Motor Angular Velocity
ω_{SYS}	Coupled Motor Shaft Angular Velocity
ω_{Vane}	Thrust Vane Angular Velocity
ω_0	Filter Corner Frequency
$\dot{\omega}_{\mathrm{SYS}}$	Coupled Motor Shaft Angular Acceleration
$\dot{\omega}_{\mathrm{Vane}}$	Thrust Vane Angular Acceleration
$\sigma_{\mathrm{DT}}(t)$	Drive Motor Torque Control Loop Integral Error Term
$\sigma_{\mathrm{DP}}(t)$	Drive Motor Position Control Loop Integral Error Term
$\sigma_{\mathrm{LT}}(t)$	Load Motor Torque Control Loop Integral Error Term
τ_{DP}	Drive Motor Position Control Loop Time Constant
τ_{DT}	Drive Motor Torque Control Loop Time Constant
τ_{LT}	Load Motor Torque Control Loop Time Constant
λ_{DP}	Drive Motor Position Control Loop Bandwidth
λ_{DT}	Drive Motor Torque Control Loop Bandwidth
λ_{LT}	Load Motor Torque Control Loop Bandwidth

SUMMARY

Missiles serve many purposes in the modern world and their control poses many complex problems. Flight paths must be followed by autopilots while position commands must be tracked by inner control loops in order to implement the missile's method of thrust vector control (TVC). There are many different methods of TVC that can be used, but for our research, exhaust flow deflection using thrust vanes was chosen. In our application, the location and material of the thrust vanes makes them susceptible to degradation due to heat and pressure. This adds additional disturbances to the system that must be addressed and characterized. Through extensive modeling and simulation, we are able to reduce the large number of disturbance variables to the most critical one: the location of the center of pressure. The force acting around this point can cause significant increases in settling time, or even instabilities, as it moves farther from the rotational axis of the thrust vane.

By designing a system consisting of a position controller surrounding a torque controller, we can accurately and robustly command the thrust vane to a desired position. We can also treat the load due to the thrust vane as a disturbance acting on the system. This method simplifies the controller design and allows us more freedom when simulating the degradation effects. Once all of the system dynamics are modeled and the controllers are designed, a hardware-in-the-loop (HWIL) setup will be used to verify simulation models and controller response. This setup was designed for future HWIL use on full missile flight paths using detailed thrust vane models to qualify flight hardware. Once the hardware is complete, it will be tested against the simulation code. This combined simulation and HWIL setup allows us to explore how a change in the location of the center of pressure affects the response of the position controller and determines the steps necessary to implement further improvements in the test setup.

CHAPTER 1
INTRODUCTION

Missiles have many applications in the modern age; both can be used for scientific discovery and for national defense. In both of these applications, the control of the missile presents a complex problem. Flight paths must be generated and followed, and the control method must be able to quickly, accurately, and robustly follow commands from the autopilot.

To date, three methods of thrust vector control (TVC) have been developed: mechanical nozzle manipulation, secondary fluid injection, and exhaust flow deflection. Mechanical nozzle manipulation yields high control authority and efficiency but requires heavy actuators and mechanical mounting. Secondary fluid injection weighs less but requires expensive pumps and nozzle fixtures for operation. This leaves exhaust flow deflection as a lighter and less expensive option. However, using exhaust flow deflection places the control surfaces in a high heat and high pressure environment, causing them to degrade during flight. The changing fin characteristics affect the load seen by the position control motor and degrade the controller's expected response to command inputs.

By examining this problem in detail, we will characterize the range of fin degradation values for which the thrust vane position controller can still meet design specifications. This characterization will be done by designing a model of the overall system in Mathwork's MATLAB® software. The models will then be used in a hardware-in-the-loop (HWIL) setup using physical missile parts and custom circuitry.

1.1 Literature Study

Before this research began, a literature study was conducted to compare different methods of TVC to better understand the physics behind it. TVC is the act of redirecting the exhaust

from a jet or rocket engine in any direction other than the axial direction [1]. In our specific case, this change in thrust direction allows for the control of a missile in all three axes: roll, pitch, and yaw. TVC also decreases the effective minimum area of engagement of a missile system. The minimum area of engagement is defined as the radial distance the missile must travel before the missile has accelerated to speeds high enough for the aerodynamic control surfaces to function [1]. Essentially, this means that the use of TVC enables the control surfaces to provide high control authority even at low air speeds. This is a critical property of TVC that allows the missile to navigate at these low speeds with high angles of attack. As outlined above, TVC can be achieved using mechanical nozzle manipulation, secondary fluid injection, or exhaust flow deflection.

1.1.1 Mechanical Nozzle Manipulation

Mechanical nozzle manipulation, also known as gimbaled thrust, is the most common form of TVC but requires powerful motors and a gimbaled mount in order to point the heavy rocket motor nozzle in the required direction. By pointing the rocket nozzle in a particular direction, the exhaust is steered in that same direction. Figure 1.1 shows an example of one of the largest gimbaled rocket engines ever built, the F-1 Saturn V rocket engine, which is the rocket that took U.S. astronauts to the moon in the 1960s. In modern applications, fighter jets such as the U.S. Air Force F-22 and F-35 utilize mechanical nozzle manipulation in order to achieve greater mobility [1].

Significant research has been conducted as to where in the nozzle geometry it is most efficient to turn the exhaust stream. The engine nozzle can be separated and gimbaled in a variety of different locations along its body. These locations correspond to different regions in the exhaust stream and have different advantages and disadvantages. Current methods have found that turning the majority of the exhaust in the subsonic region of the nozzle yields the highest efficiency [3]. This subsonic region occurs shortly before the nozzle throat, shown in Figure 1.2.

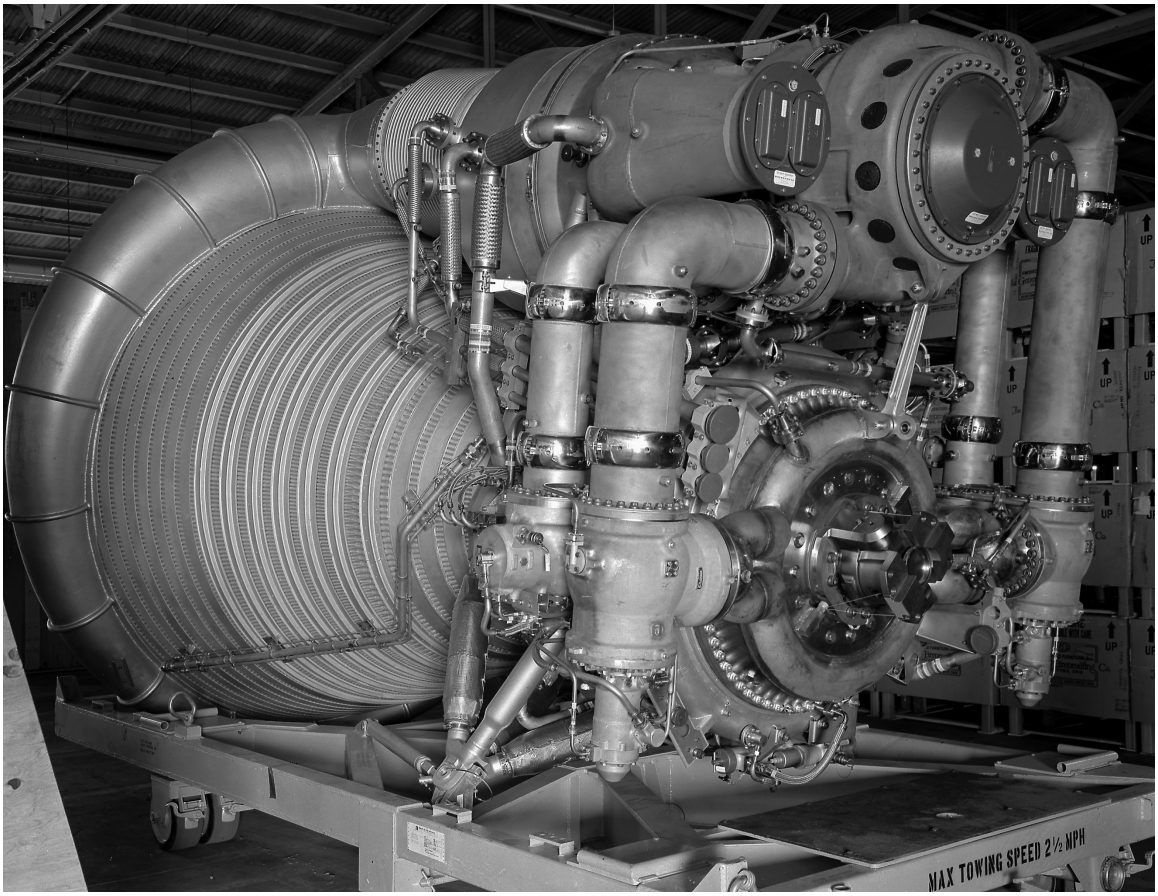


Figure 1.1: Saturn V F-1 Gimbaled Rocket Engine [2]

1.1.2 Secondary Fluid Injection

Secondary fluid injection functions in a similar manner to mechanical nozzle manipulation in that the nozzle dynamics are changed in order to steer the exhaust. By injecting a second fluid such as Freon or using bleed air from an engine compressor, artificial nozzle boundaries can be created and the engine exhaust can be modified and steered. This method often weighs less than gimbaled nozzles as they can be constructed using fixed geometry and do not need complex adjustable hardware.

Currently three different methods of secondary fluid injection exist: counter flow, shock vector control, and throat shifting. Each of these techniques have differing degrees of efficiency and effective vectoring angles [4]. Counter flow functions by using suction in

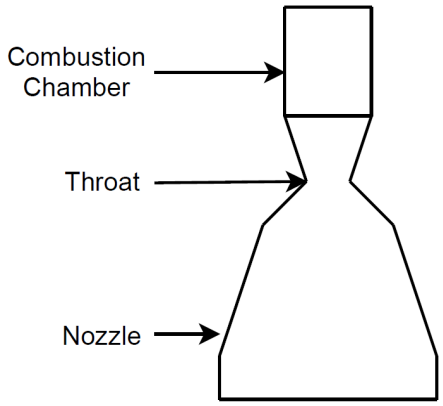


Figure 1.2: Rocket Engine Geometry

the aft portion of the nozzle, creating a reverse flow along the engine nozzle collar. This results in a drop in pressure and therefore an increase in velocity near the reverse flow. This method provides the highest level of efficiency. Shock vector control functions by creating shock barriers in the supersonic region of the exhaust flow. This method trades high effective vectoring range for lowered efficiency. Finally, throat shifting works in the subsonic region of the engine exhaust flow, as in mechanical nozzle manipulation. This method, however, suffers from both low efficiency and low performance [5].

1.1.3 Exhaust Flow Deflection

Exhaust flow deflection is accomplished through the use of smaller fins or "vanes" placed on the interior of the rocket nozzle and in the path of the engine thrust vector. By rotating these fins, one can effectively steer the rocket thrust, creating torque about the rocket's center of mass, turning the rocket. Exhaust flow deflection was most famously implemented in the 1940s on the German V2 rocket by using a series of internal graphite paddles [6]. More recently this method of TVC has been achieved using external paddles, or "post-exit vanes," that can be pushed into the exhaust path, as in the case of the Rockwell-MBB X-31 experimental jet fighter [1]. The thrust vane method of the V2 rocket and the paddle

method of the X-31 are shown in Figure 1.3 and Figure 1.4 respectively.

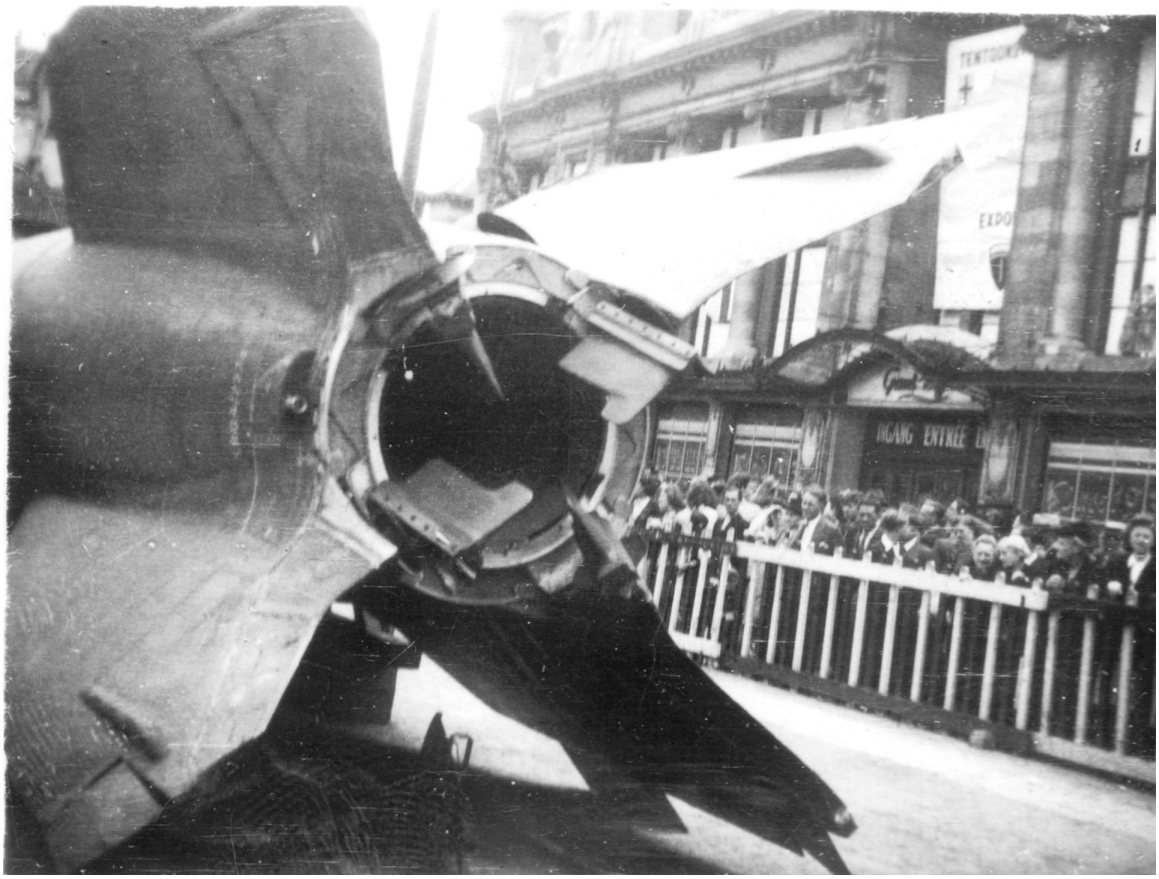


Figure 1.3: V2 Rocket Thrust Vanes [7]

1.2 Problem Statement

For this research, exhaust flow deflection through the use of thrust vanes was chosen as the TVC method. This decision was made because the large gimbals and fluid injection systems required for mechanical nozzle manipulation and secondary fluid injection would raise the cost and weight of the missile substantially. Therefore, a small thrust vane has been designed to be used on this project. As discussed previously, thrust vanes are exposed to intense heat and pressure. This exposure is generally addressed by making the thrust vanes out of materials such as graphite or ceramics which can withstand these extreme conditions. However, the problem with these materials is the high manufacturing cost. For

Figure 1.4: Rockwell X-31 TVC Paddles [8]

this theoretical problem those high costs are unacceptable and so less ideal materials were used, leaving us with thrust vanes whose properties are expected to change during flight. Therefore, the question is raised: for what range of thrust vane degradation can we design an accurate and robust controller?

In order to design this controller, we must understand how the loading effects of the thrust vane are generated, design a controller that will perform to the standards outlined in the project, and characterize the range of degradation that can be compensated for by the controller. Once we have designed a satisfactory controller, the determination can be made if the range of degradation that can be handled is within the expected degradation boundaries, which are yet to be defined.

CHAPTER 2

MODELING

This research will focus on the missile's ability to command its control surfaces and the design of the thrust vane position controller. In order to explore this area, we must first generate a system in which the load due to the degrading thrust vane can be emulated and the controller can be designed. Our ideal physical system is shown in Figure 2.1. Here, we

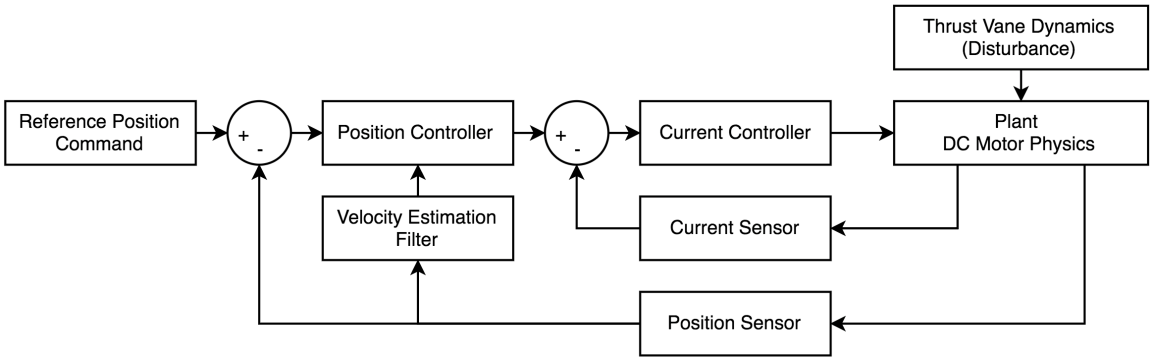


Figure 2.1: Ideal System Diagram

treat the thrust vane dynamics, outlined below, as a disturbance to the ideal motor position controller. This physical disturbance is simple to implement in software but is hard to implement in a HWIL setup. Therefore, the system was changed to allow us to emulate the physical disturbance of the degrading thrust vanes on the actual project hardware. To do this in simulation, we sum the torque outputs of two direct current (DC) motor electrical dynamics blocks, using one as the actual missile thrust vane drive motor and the other to emulate the thrust vane load, and have them act upon a single mechanical load. In hardware this is accomplished by mechanically coupling the output shafts of the two motors. This is shown in Figure 2.2. This chapter will show the process in which the DC motor models were generated, highlight how the load model was defined using physics, and discuss the hypothesized effects this load will have on the controller.

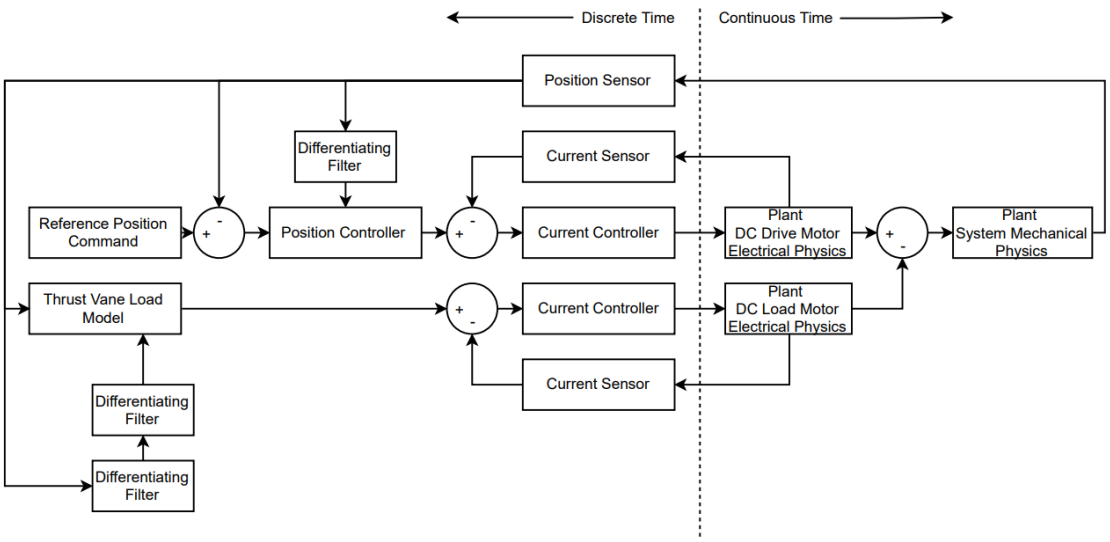


Figure 2.2: System Diagram with Load Motor

2.1 Plant Models

As seen in Figure 2.2, our plant consists of two DC motors that have been mechanically coupled. Each motor consists of electrical dynamics and mechanical dynamics, but due to their coupling, our system consists of two electrical plants and one mechanical plant. These three plants must be modeled for simulation and controller design purposes. We can begin by modeling the DC motor electrical dynamics as a function of motor inductance, L, motor resistance, R, motor torque constant, k_{t}, terminal voltage, V, and motor rotational velocity, ω. Since both the drive and load motors have the same dynamics, we will append the subscript $_{\mathrm{D}}$ to signify the drive motor and $_{\mathrm{L}}$ to signify the load motor. This gives us the following two differential equations:

$$L_{\mathrm{D}}\dot{I}_{\mathrm{D}} = V_{\mathrm{D}} - R_{\mathrm{D}}I_{\mathrm{D}} - k_{\mathrm{tD}}\omega_{\mathrm{D}} \tag{2.1}$$

and

$$L_L\dot{I}_{\mathrm{L}} = V_{\mathrm{L}} - R_{\mathrm{L}}I_{\mathrm{L}} - k_{\mathrm{tL}}\omega_{\mathrm{L}} \tag{2.2}$$

We can choose motor currents I_{D} and I_{L} as our state variables $x_{\mathrm{eD}}(t)$ and $x_{\mathrm{eL}}(t)$, terminal voltages, $V_{\mathrm{D}}(t)$ and $V_{\mathrm{L}}(t)$, as our inputs $u_{\mathrm{eD}}(t)$ and $u_{\mathrm{eL}}(t)$, motor torques acting on the shaft, T_{D} and T_{L}, as our outputs $y_{\mathrm{eD}}(t)$ and $y_{\mathrm{eL}}(t)$, motor rotational velocity ω_{D} and ω_{L} as disturbances $w_{\mathrm{eD}}(t)$ and $w_{\mathrm{eL}}(t)$, and re-write Equations 2.1 and 2.2 in state space form as

$$\dot{x}_{\mathrm{eD}}(t) = \left[-\frac{R_{\mathrm{D}}}{L_{\mathrm{D}}}\right] x_{\mathrm{eD}}(t) + \left[\frac{1}{L_{\mathrm{D}}}\right] u_{\mathrm{eD}}(t) + \left[-\frac{k_{\mathrm{tD}}}{L_{\mathrm{D}}}\right] w_{\mathrm{eD}}(t) \tag{2.3a}$$

$$y_{\mathrm{eD}}(t) = \left[k_{\mathrm{tD}}\right] x_{\mathrm{eD}}(t) \tag{2.3b}$$

and

$$\dot{x}_{\mathrm{eL}}(t) = \left[-\frac{R_{\mathrm{L}}}{L_{\mathrm{L}}}\right] x_{\mathrm{eL}}(t) + \left[\frac{1}{L_{\mathrm{L}}}\right] u_{\mathrm{eL}}(t) + \left[-\frac{k_{\mathrm{tL}}}{L_{\mathrm{L}}}\right] w_{\mathrm{eL}}(t) \tag{2.4a}$$

$$y_{\mathrm{eL}}(t) = \left[k_{\mathrm{tL}}\right] x_{\mathrm{eL}}(t) \tag{2.4b}$$

using the subscript $_{\mathrm{e}}$ to signify the electrical dynamics.

As stated before, since the motors are mechanically coupled, there is a single mechanical plant for this system. We can express the mechanical dynamics of the two-motor system as a function of system inertia, J_{SYS}, the rotational velocity of the system, ω_{SYS}, the friction co-efficient of the system, b_{SYS}, and the total torque acting on the system, T_{SYS}.

$$J_{\mathrm{SYS}}\dot{\omega}_{\mathrm{SYS}} = T_{\mathrm{SYS}} - b_{\mathrm{SYS}}\omega_{\mathrm{SYS}} \tag{2.5}$$

Equation 2.5 can be represented in state space form by choosing our state variables, $x_{\mathrm{m}}(t)$, as the column vector of system angle θ_{SYS} and angular velocity ω_{SYS}, or

$$x_{\mathrm{m}}(t) = \begin{bmatrix} \theta_{\mathrm{SYS}} \\ \omega_{\mathrm{SYS}} \end{bmatrix}$$

Furthermore, choosing our output, $y_{\mathrm{m}}(t)$, as θ_{SYS}, and our input, $u_{\mathrm{SYS}}(t)$, as T_{SYS} we can

express the motor mechanical dynamics in state space form as

$$\dot{x}_{\mathrm{m}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{b_{\mathrm{SYS}}}{J_{\mathrm{SYS}}} \end{bmatrix} x_{\mathrm{m}}(t) + \begin{bmatrix} 0 \\ \dfrac{1}{J_{\mathrm{SYS}}} \end{bmatrix} u_{\mathrm{SYS}}(t) \tag{2.6a}$$

$$y_{\mathrm{m}}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_{\mathrm{m}}(t) \tag{2.6b}$$

In addition, due to the mechanical coupling of the system, we can define the total torque acting on the system, T_{SYS}, as the subtraction of the torques generated by each motor, or

$$T_{\mathrm{SYS}} = T_{\mathrm{DS}} - T_{\mathrm{LS}} \tag{2.7}$$

or

$$u_{\mathrm{SYS}}(t) = y_{\mathrm{eD}}(t) - y_{\mathrm{eL}}(t) \tag{2.8}$$

Equations 2.3a – 2.4b and 2.6a – 2.8 give us the full state space representation of our combined electrical and mechanical system.

2.2 Thrust Vane Load Model

Before we can evaluate the effects that vane degradation has on the response of the position controller, we must first create an accurate model of the loading forces acting on the thrust vane. By looking at the forces acting on the vane, we can generate a model of the load in terms of variables we can test. This section will detail the process of creating the thrust vane load model and introduce the effects of thrust vane degradation.

2.2.1 Physics of the Thrust Vane Load

In order to create the thrust vane load model, let us examine the forces acting on the thrust vane. Figure 2.3 shows a simple thrust vane design and the locations of the center of gravity, CG, and the center of pressure, CP. There are three different torques acting on the

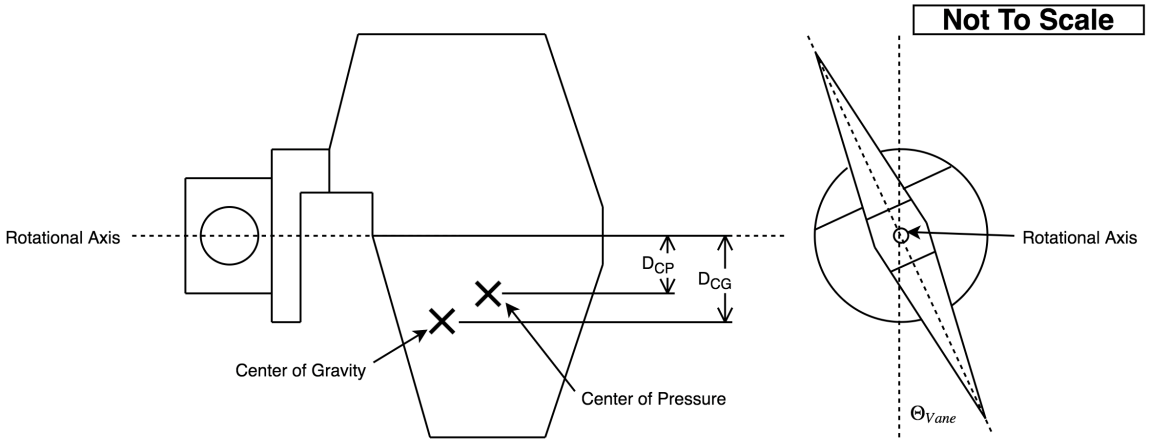


Figure 2.3: Thrust Vane Example

thrust vane: torque due to the mass properties, T_{CG}, which acts about the CG, torque due to aerodynamic forces, T_{CP}, which acts about the CP, and the torque due to the inertia of the thrust vane, T_{J}. T_{CG} is caused by the force due to the acceleration, F_{CG}, of the rocket body acting on the thrust vane on the CG at a distance D_{CG} from the rotational axis, or

$$T_{\mathrm{CG}} = F_{\mathrm{CG}} \times D_{\mathrm{CG}}$$

In addition, using Newton's Second Law of Motion

$$F = ma$$

and the definition of a cross product

$$A \times B = \|A\|\|B\| \cos(\phi)$$

we can relate the force due to the acceleration of the rocket to the thrust vane's mass, m_V, thrust vane angle, θ_{Vane}, and rocket body acceleration, a_R, as

$$T_{\mathrm{CG}} = m_{\mathrm{V}} \|a_{\mathrm{R}}\| \, D_{\mathrm{CG}} \cos(\theta_{\mathrm{Vane}}) \tag{2.9}$$

In the same fashion, T_{CP} is caused by the aerodynamic forces, F_{CP}, acting on the CP at a distance D_{CP} from the rotational axis. Applying the same method as above gives us the equation

$$T_{\mathrm{CP}} = F_{\mathrm{CP}} \times D_{\mathrm{CP}}$$

F_{CP} can be calculated using the peak thrust of the rocket motor, F_{RM}, the thrust vane surface area, A_{V}, and the rocket nozzle area, A_{RM}, using

$$\frac{F_{\mathrm{RM}}}{A_{\mathrm{RM}}} = -p = \frac{F_{\mathrm{CP}}}{A_{\mathrm{V}}}$$

where p is pressure. We can solve this equation for F_{CP}.

$$F_{\mathrm{CP}} = \frac{A_{\mathrm{V}}}{A_{\mathrm{RM}}} F_{\mathrm{RM}} \tag{2.10}$$

Therefore,

$$T_{\mathrm{CP}} = \frac{A_{\mathrm{V}}}{A_{\mathrm{RM}}} \|F_{\mathrm{RM}}\| \, D_{\mathrm{CP}} \cos(\theta_{\mathrm{Vane}}) \tag{2.11}$$

Finally, the torque due to the inertia, T_{J}, depends solely on the vane inertia, J_{V}, and the angular acceleration of the thrust vane, $\dot{\omega}_{\mathrm{Vane}}$,

$$T_{\mathrm{J}} = J_{\mathrm{V}} \dot{\omega}_{\mathrm{Vane}} \tag{2.12}$$

As a result, the total load torque, T_{L}, is defined as the sum of Equations 2.9, 2.11, and 2.12,

$$T_{\mathrm{L}} = T_{\mathrm{CG}} + T_{\mathrm{CP}} + T_{\mathrm{J}} \tag{2.13}$$

or

$$T_{\mathrm{L}} = m_{\mathrm{V}} \|a_{\mathrm{R}}\| \, D_{\mathrm{CG}} \cos(\theta_{\mathrm{Vane}}) + \frac{A_{\mathrm{V}}}{A_{\mathrm{RM}}} \|F_{\mathrm{RM}}\| \, D_{\mathrm{CP}} \cos(\theta_{\mathrm{Vane}}) + J_{\mathrm{V}} \dot{\omega}_{\mathrm{Vane}} \tag{2.14}$$

Equation 2.14 gives us a model of the load with respect to some known constant parameters such as m_{V}, a_{R}, and F_{RM} as well as variable parameters such as D_{CG}, D_{CP}, $\dot{\omega}_{\mathrm{Vane}}$, and θ_{Vane}.

2.2.2 Thrust Vane Degradation

Before exploring the loading effects on the controller, we must create a theoretical missile and thrust vane. For this research, we are using a theoretical rocket engine with a peak thrust of $F_{\mathrm{RM}} = 9000$ N, peak acceleration of $a_{\mathrm{R}} = 15\ \mathrm{m/s^2}$, and nozzel area of $A_{\mathrm{RM}} = 145931.754\ \mathrm{mm^2}$. We also have a thrust vane of nominal mass $m_{\mathrm{V}} = 0.1$ g and nominal surface area $A_{\mathrm{V}} = 250\ \mathrm{mm^2}$. This gives us $F_{\mathrm{CP}} = 15.42$ N using Equation 2.10. The location of D_{CP} is initially at 0 mm and the location of D_{CG} is at 5 mm.

Equation 2.11 clearly shows that the load on the motor depends greatly on D_{CP} due to the high expected value of F_{RM}. In order to eliminate this load, the ideal thrust vane is designed with the center of pressure located at the rotational axis of the vane, making $D_{\mathrm{CP}} = 0$. However, the center of pressure is expected to change as the fin degrades, causing D_{CP} to change proportionally. Therefore, we must determine the range of values of D_{CP} for which the controller will still adequately function and see what effects this has on our control authority.

Similarly, Equation 2.9 shows that the torque due to the center of mass will vary as D_{CG} increases. Due to the relatively small values of m_{V} and a_{R} we can safely conclude that the torque due to the CG plays a minimal part in the loading of the system. This research will therefore focus on the effects of changing the location of the CP by categorizing the range of D_{CP} values for which the thrust vane position controller still functions.

2.3 Hypothesized Effect on the Position Controller

We hypothesize that the load will have a drastically different effect on the controller based on how the center of pressure drifts. The worst case scenario would be if the center of pres-

sure moves perpendicularly to the axis of rotation, causing D_{CP} to increase most rapidly. If the center of pressure moves in a positive direction (moving in the direction of the rocket thrust, D_{CP} becoming positive) we expect to see extended settling times due to T_{CP} fighting against the controller's effort. If the center of pressure moves in a negative direction (moving against the direction of the rocket thrust, D_{CP} becoming negative) we expect to see increased overshoot and ringing due to T_{CP} acting with the controller's effort.

CHAPTER 3
CONTROLLER DESIGN

In order create a robust and stable position controller, an accurate model of the desired control system must first be generated. For this research, an internal torque control loop with an outer position control loop was chosen, shown in Figure 2.2. The indirect controller design method was chosen to design these controllers. This method does not explicitly take the micro-controller's interrupt service routine (ISR) update rate into account when calculating gains. Instead it relies on the control engineer to make sure that the bandwidth of the controller is substantially less than the update rate of the ISR.

3.1 Position Controller

With the plant modeled, we move on to the design of the controllers themselves, beginning with the outer position control loop. The design of the thrust vane position controller depends on the mechanical dynamics of the DC motor from Equation 2.6a. For this controller, we are using the subscript $_{\mathrm{DP}}$ to designate the drive motor position controller. Adding state space integral control to the existing state space representation of the mechanical system allows us to define

$$u_{\mathrm{m}}(t) = -\begin{bmatrix} K_{11\mathrm{DP}} & K_{12\mathrm{DP}} \end{bmatrix} x_{\mathrm{m}}(t) - K_{2\mathrm{DP}}\sigma_{\mathrm{DP}}(t) \tag{3.1a}$$

$$\dot{\sigma}_{\mathrm{DP}}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x_m(t) - r_{\mathrm{DP}}(t) \tag{3.1b}$$

where r_{DP} is the drive motor position control loop reference command input, σ_{DP} is the drive motor position control loop integral error term, $K_{11\mathrm{DP}}$, $K_{12\mathrm{DP}}$, and $K_{2\mathrm{DP}}$ are the gains associated with the drive motor position control loop, and u_{m} is the state space input generated by the integral controller. Substituting Equations 3.1a and 3.1b into Equation

2.6a and combining like terms give us the state space representation of the position controller:

$$\begin{bmatrix} \dot{x}_{\mathrm{m}}(t) \\ \dot{\sigma}_{\mathrm{DP}}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\dfrac{K_{11\mathrm{DP}}}{J_{\mathrm{SYS}}} & -\dfrac{K_{12\mathrm{DP}}}{J_{\mathrm{SYS}}} & -\dfrac{K_{2\mathrm{DP}}}{J_{\mathrm{SYS}}} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_m(t) \\ \sigma_{\mathrm{DP}}(t) \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} r_{\mathrm{DP}}(t) - \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} w_{\mathrm{DP}}(t) \quad (3.2)$$

where $w_{\mathrm{DP}}(t)$ is the effect due to the physical thrust vane load. During design, we treat this load, defined in Equation 2.14, as a disturbance input to the system when calculating gains. From state space control theory, we know that the stability of the control loop depends on the eigenvalues of the modified A matrix

$$\widehat{A}_{\mathrm{DP}} = \begin{bmatrix} 0 & 1 & 0 \\ -\dfrac{K_{11\mathrm{DP}}}{J_{\mathrm{SYS}}} & -\dfrac{K_{12\mathrm{DP}}}{J_{\mathrm{SYS}}} & -\dfrac{K_{2\mathrm{DP}}}{J_{\mathrm{SYS}}} \\ 1 & 0 & 0 \end{bmatrix} \quad (3.3)$$

To determine the locations of the eigenvalues of this matrix, we must first find the characteristic polynomial, or

$$\det(sI - \widehat{A}_{\mathrm{DP}}) = s^3 + \frac{K_{12\mathrm{DP}}}{J_{\mathrm{SYS}}}s^2 + \frac{K_{11\mathrm{DP}}}{J_{\mathrm{SYS}}}s + \frac{K_{2\mathrm{DP}}}{J_{\mathrm{SYS}}} \quad (3.4)$$

where I is the identity matrix. We can guarantee stability by placing all eigenvalues of this matrix at the same point on the negative real axis in the complex plane. This also allows us to define τ_{DP}, the drive position control loop time constant, for this control loop. This

yields

$$(s + \lambda_{\mathrm{DP}})^3 = s^3 + 3\lambda_{\mathrm{DP}}s^2 + 3\lambda_{\mathrm{DP}}^2 s + \lambda_{\mathrm{DP}}^3 \tag{3.5}$$

where

$$\lambda_{\mathrm{DP}} = \frac{1}{\tau_{\mathrm{DP}}} \tag{3.6}$$

and is the desired position of all the eigenvalues. Setting the characteristic polynomial in Equation 3.4 equal to our desired polynomial in Equation 3.5 and solving the resulting system of equations give us the following gain calculations:

$$K_{11\mathrm{DP}} = 3J_{\mathrm{SYS}}\lambda_{\mathrm{DP}}^2 \tag{3.7a}$$

$$K_{12\mathrm{DP}} = 3J_{\mathrm{SYS}}\lambda_{\mathrm{DP}} \tag{3.7b}$$

$$K_{2\mathrm{DP}} = J_{\mathrm{SYS}}\lambda_{\mathrm{DP}}^3 \tag{3.7c}$$

We now have equations directly relating the desired bandwidth for the outer loop to the controller gains. This gain calculation will guarantee stability of the outer position control loop under nominal loads and adequate ISR update rates.

3.2 Torque Controllers

While the design of the position controller depends completely on the mechanical dynamics of the system, the design of both inner torque control loops focuses solely on the DC motor electrical dynamics in Equation 2.3a and 2.4a. For the drive motor torque controller, we are using the subscript $_{\mathrm{DT}}$. For the sake of clarity and brevity, we will detail the design of the drive torque control loop only. It should be understood that the load torque control loop follows the exact same design steps and will be shown at the end of this section, the only difference being a change in subscript. Adding state space integral control to the existing

state space representation of the drive motor electrical system allows us to define

$$u_{\mathrm{eD}}(t) = -K_{1\mathrm{DT}}k_{\mathrm{tD}}x_{\mathrm{eD}}(t) - K_{2\mathrm{DT}}\sigma_{\mathrm{DT}}(t) \tag{3.8a}$$

$$\dot{\sigma}_{\mathrm{DT}}(t) = k_{\mathrm{tD}}x_{\mathrm{eD}}(t) - r_{\mathrm{DT}}(t) \tag{3.8b}$$

where r_{DT} is the drive motor torque control loop reference command input (generated by the outer loop), σ_{DT} is the drive motor torque control loop integral error term, and $K_{1\mathrm{DT}}$ and $K_{2\mathrm{DT}}$ are the gains associated with the drive motor torque control loop. Substituting Equations 3.8a and 3.8b into Equation 2.3a and combining like terms give us the state space representation of the torque controller

$$\begin{bmatrix} \dot{x}_{\mathrm{eD}}(t) \\ \dot{\sigma}_{\mathrm{DT}}(t) \end{bmatrix} = \begin{bmatrix} -\dfrac{R_{\mathrm{D}} + (K_{1\mathrm{DT}}k_{\mathrm{tD}})}{L_{\mathrm{D}}} & -\dfrac{K_{2\mathrm{DT}}}{L_{\mathrm{D}}} \\ k_{\mathrm{tD}} & 0 \end{bmatrix} \begin{bmatrix} x_{\mathrm{eD}}(t) \\ \sigma_{\mathrm{DT}}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} r_{\mathrm{DT}}(t) + \begin{bmatrix} -\dfrac{k_{\mathrm{tD}}}{L_{\mathrm{D}}} \\ 0 \end{bmatrix} w_{\mathrm{DT}}(t) \tag{3.9}$$

Again, we know that the stability of the control loop depends on the eigenvalues of the modified A matrix

$$\widehat{A}_{\mathrm{DT}} = \begin{bmatrix} -\dfrac{R_{\mathrm{D}} + (K_{1\mathrm{DT}}k_{\mathrm{tD}})}{L_{\mathrm{D}}} & -\dfrac{K_{2\mathrm{DT}}}{L_{\mathrm{D}}} \\ k_{\mathrm{tD}} & 0 \end{bmatrix} \tag{3.10}$$

for which we must again find the characteristic polynomial in order to place the eigenvalues. This polynomial is defined as

$$\det(sI - \widehat{A}_{\mathrm{DT}}) = s^2 + \frac{R_{\mathrm{D}} + (K_{1\mathrm{DT}}k_{\mathrm{tD}})}{L_{\mathrm{D}}}s + \frac{K_{2\mathrm{DT}}}{L_{\mathrm{D}}} \tag{3.11}$$

where I is the identity matrix. As done in the position controller, we can guarantee stability by placing all eigenvalues of $\widehat{A}_{\mathrm{DT}}$ at the same point on the negative real axis in the complex

plane. Again, we define τ_{DT}, the drive torque control loop time constant, and

$$(s + \lambda_{\mathrm{DT}})^2 = s^2 + 2\lambda_{\mathrm{DT}}s + \lambda_{\mathrm{DT}}^2 \tag{3.12}$$

where

$$\lambda_{\mathrm{DT}} = \frac{1}{\tau_{\mathrm{DT}}} \tag{3.13}$$

and is the desired position of all the eigenvalues. Setting the characteristic polynomial in Equation 3.11 equal to our desired polynomial in Equation 3.12 and solving the resulting system of equations gives us the following gain calculations:

$$K_{1\mathrm{DT}} = \frac{2L_{\mathrm{D}}\lambda_{\mathrm{DT}} - R_{\mathrm{D}}}{k_{\mathrm{tD}}} \tag{3.14a}$$

$$K_{2\mathrm{DT}} = \frac{L_{\mathrm{D}}\lambda_{\mathrm{DT}}^2}{k_{\mathrm{tD}}} \tag{3.14b}$$

As stated above, the design of the load torque controller follows the same steps as those detailed for the drive torque controller. Therefore we know that the state space representation of the load torque controller is

$$\begin{bmatrix} \dot{x}_{\mathrm{eL}}(t) \\ \dot{\sigma}_{\mathrm{LT}}(t) \end{bmatrix} = \begin{bmatrix} -\dfrac{R_{\mathrm{L}} + (K_{1\mathrm{LT}}k_{\mathrm{tL}})}{L_{\mathrm{L}}} & -\dfrac{K_{2\mathrm{LT}}}{L_{\mathrm{L}}} \\ k_{\mathrm{tL}} & 0 \end{bmatrix} \begin{bmatrix} x_{\mathrm{eL}}(t) \\ \sigma_{\mathrm{LT}}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} r_{\mathrm{LT}}(t) + \begin{bmatrix} -\dfrac{k_{\mathrm{tL}}}{L_{\mathrm{L}}} \\ 0 \end{bmatrix} w_{\mathrm{LT}}(t) \tag{3.15}$$

and the gain calculations for the load torque controller are

$$K_{1\mathrm{LT}} = \frac{2L_{\mathrm{L}}\lambda_{\mathrm{LT}} - R_L}{k_{\mathrm{tL}}} \tag{3.16a}$$

$$K_{2\mathrm{LT}} = \frac{L_{\mathrm{L}}\lambda_{\mathrm{LT}}^2}{k_{\mathrm{tL}}} \tag{3.16b}$$

We now have equations directly relating the desired bandwidth for the inner loop to the controller gains.

3.3 Differentiating Filter

In order to properly control the position of the drive motor, we must be able to observe both the position and velocity of the system. This is done by reading in the encoder values to a digital differentiation filter to get velocity. In the same way, the load model requires both thrust vane position and thrust vane acceleration to function. By passing the vane position through two differentiation filters, we can obtain a value for vane angular acceleration. Figure 2.2 shows the differentiation filters being fed by the motor position sensors. The design for these filters was taken from "Novel Approach to Designing Digital Differentiators" by M. A. Al-Alaoui [9]. By taking an ideal Tick integrator of the form

$$G(z) = \frac{T_{\text{Tick}}(0.3585z^2 + 1.2832z + 0.3584)}{(z^2 - 1)} \tag{3.17}$$

with a sample period of T_{Tick} seconds, taking its inverse and replacing the now unstable pole at $z = -3.2750$ with a pole at $z = -\dfrac{1}{-3.2750} = -0.30534$ we get

$$H(z) = A_{\text{VE}} \frac{(z^2 - 1)}{(z^2 + 0.611z + 0.0932)} \tag{3.18}$$

where the filter constant $A_{\text{VE}} = \dfrac{1}{1.17376\ T_{\text{VE}}}$ and T_{VE} is the sample period of the differentiation filter in seconds. We can see in Figure 3.1 that the filter approximates the magnitude of an ideal differentiator up to around half of the sample frequency. In addition, the phase of the filter remains linear over this entire range.

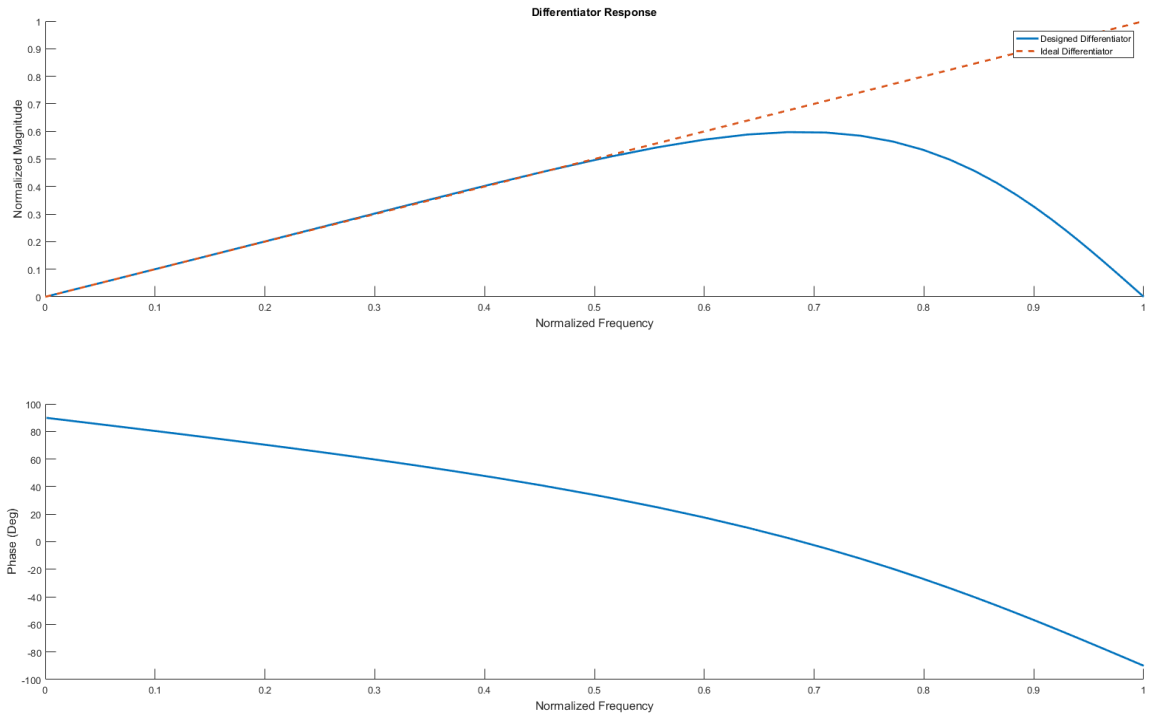


Figure 3.1: Amplitude and Phase Response of Differentiation Filter

CHAPTER 4
SYSTEM DESIGN

During the research and development process, HWIL is often used to validate hardware. This allows us an opportunity to check for hardware errors, safety concerns, or other design mistakes that may not be present in the simulation. In addition, eventually moving from simulation to hardware is guaranteed, as the end result is a physical product. For these reasons, HWIL testing was planned from the beginning. This chapter will outline the design of the HWIL setup that was created for this research as well as the design of the simulation and HWIL software.

4.1 Hardware Design

In order to check for design errors and parameter mismatches, verify the validity of actual missile hardware, and check the physical hardware against the models, we chose to implement our models in a HWIL setup. By using actual hardware in our HWIL setup, we can test the limits of the power converters, write code for the Texas Instruments (TI) TMS320F28069PZPQ micro-controller, determine if the ISR rate we simulated is feasible, and introduce non-ideal conditions such as friction and slip in the gearbox which are not explicitly modeled. This observation does not mean that the simulation model should be treated as completely "ideal." Some important non-idealities such as the discrete time nature of the controller and quantization of the sensors and voltage outputs are modeled.

The HWIL setup will also provide us with important information when verifying simulation models. For example, we can test the motor models by applying a known voltage to the terminals, plot the position and speed generated, and compare this to our simulation models. This section will outline the details of the HWIL setup, including motor and sensor hardware used, digital and analog circuitry design, and printed circuit board (PCB) layout.

4.1.1 Motors, Encoders, and Gearboxes

For this research, motor assemblies from Maxon where chosen due to their small size, light weight, and relatively low cost. We are using 12V DCX16CS motors with EASY16 absolute Synchronous Serial Interface (SSI) encoder assemblies. These assemblies are a three-part package consisting of an encoder, a motor, and a gearbox shown in Figure 4.1. The drive motor has a 6.6:1 gearbox and the load motor has a 3.9:1 gearbox based on the

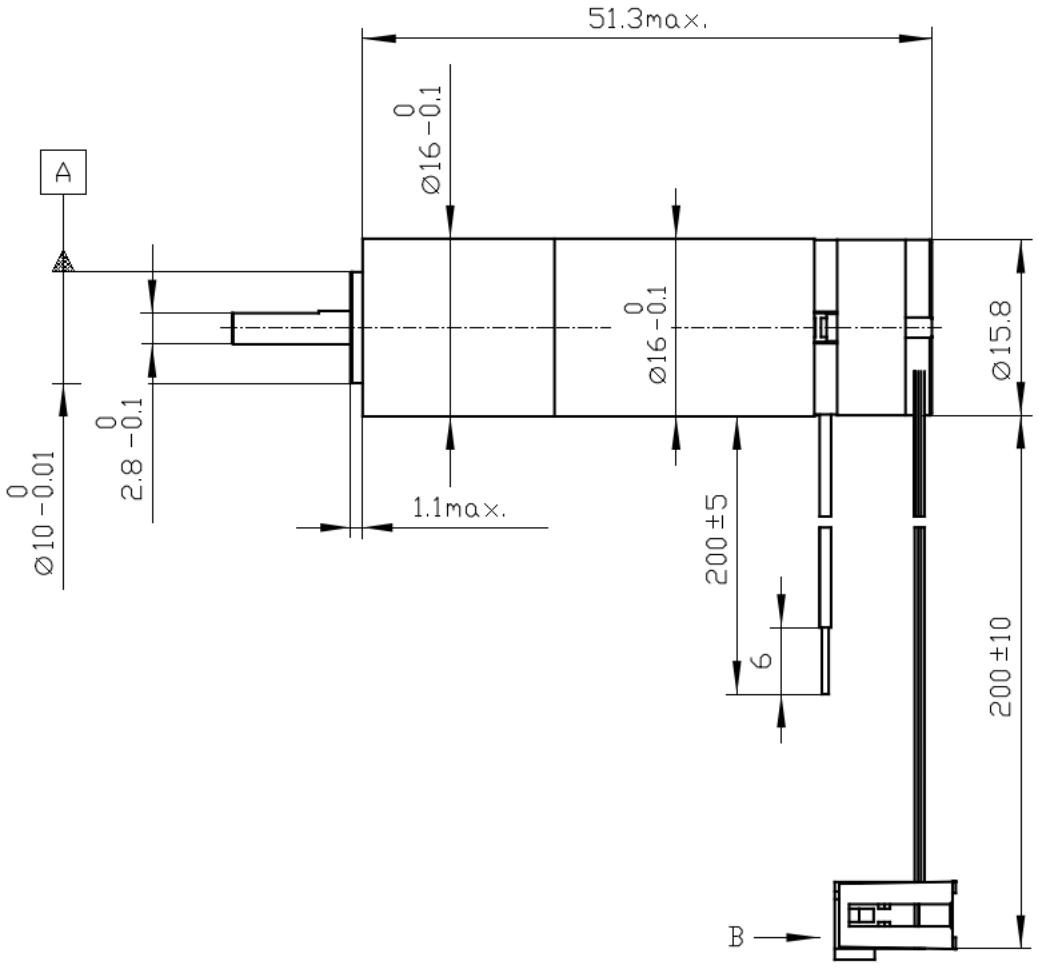


Figure 4.1: Maxon DC Motor Assembly [10]

availability of prototype hardware. Due to the gearbox, the drive motor and load motor encoder positions, θ_{D} and θ_{L}, are related to the position of the coupled motor shafts, θ_{SYS},

through their respective gear ratios, G_{D} and G_{L}.

$$\theta_{\mathrm{SYS}} = \theta_{\mathrm{D}} G_{\mathrm{D}} \tag{4.1a}$$

$$\theta_{\mathrm{SYS}} = -\theta_{\mathrm{L}} G_{\mathrm{L}} \tag{4.1b}$$

where $G_D = \dfrac{1}{6.6}$ and $G_L = \dfrac{1}{3.9}$. While the same relation holds true for the motor speed shown in Equations 4.2a and 4.2b,

$$\omega_{\mathrm{SYS}} = \omega_{\mathrm{D}} G_{\mathrm{D}} \tag{4.2a}$$

$$\omega_{\mathrm{SYS}} = -\omega_{\mathrm{L}} G_{\mathrm{L}} \tag{4.2b}$$

the opposite relation holds for the output torque from the gearbox, T_{DS} and T_{LS}, in relation to the output torque before the gearbox T_{D} and T_{L}, as shown in Equations 4.3a and 4.3a.

$$T_{\mathrm{DS}} = \frac{T_{\mathrm{D}}}{G_{\mathrm{D}}} \tag{4.3a}$$

$$T_{\mathrm{LS}} = \frac{T_{\mathrm{L}}}{G_{\mathrm{L}}} \tag{4.3b}$$

4.1.2 HWIL Schematic Design

In order to drive the motors selected, a motor controller PCB was designed. The Texas Instruments TMS320F28069PZPQ micro-controller, shown in Figure 4.2, was chosen as the sole processor for this application because it allows us to drive all eight pulse width modulation (PWM) signals required for dual motor operation, read in two motor encoders, sense two motor current wave-forms, run three state space control loops at 30kHz, run velocity and acceleration estimation filters, and control status LEDs and GPIO lines. To operate, the board accepts a 12V DC power input and generates the required 3.3V and 5V rails to be used on the board. This is done using the circuit in Figure 4.3.

One of the key features of the HWIL setup is its ability to drive two separate DC motors.

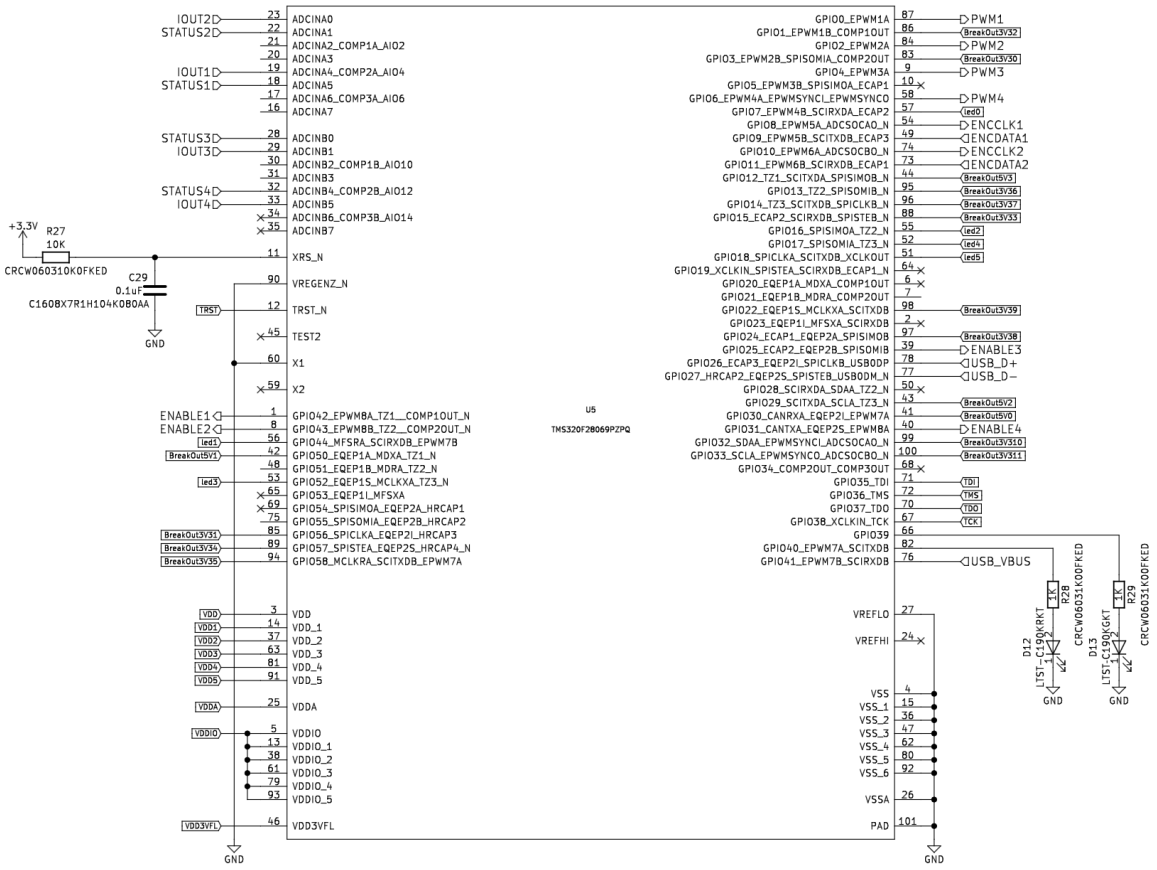


Figure 4.2: TMS320F28069PZPQ Microcontroller Circuitry

We used two identical half-bridge circuits per DC motor, for a total of four on the board. This circuit, shown in Figure 4.4, allows us to drive a motor with the full $\pm 12V$ operation required. This was done using two inverse synchronized PWM signals coming from the micro-controller per motor.

In order to implement a position controller, we must be able to read in the motor encoder in order to obtain the motor angle. The encoders discussed in Section 4.1.1 are absolute encoders which output a 5V SSI signal encoded in Gray symmetric code. This difference in voltage level means that we must first level shift the signal down from 5V to 3.3V using the circuit shown in Figure 4.5. The use of Gray code encoding adds an additional step when calculating the motor position. By converting the Gray code to a binary integer value

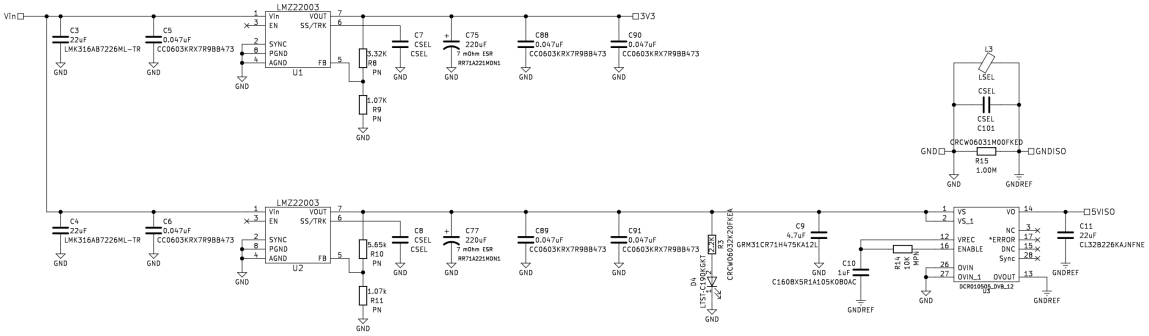


Figure 4.3: Board Power Rail Circuitry

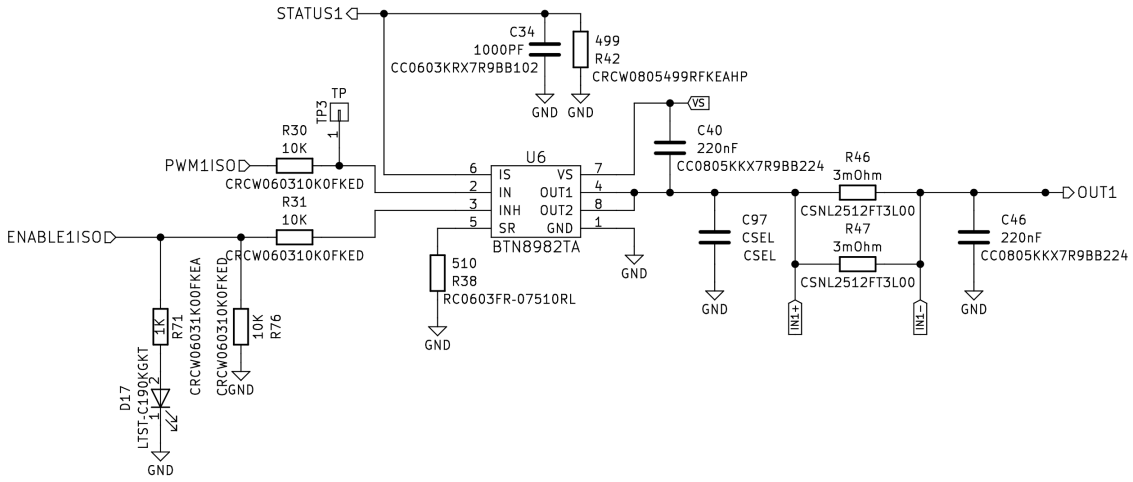


Figure 4.4: Half Bridge Motor Driver Circuitry

(N_{ENC}) in code, we can calculate the motor position, θ_{M}, in radians, as

$$\theta_{\text{M}} = \frac{2\pi}{2^{14}} N_{\text{ENC}} - \pi \tag{4.4}$$

where the offset of π was added to adjust the absolute zero point to half scale. This value can then be used in Equation 4.1a or 4.1b as needed to obtain the position of the coupled motor shaft or load motor.

We also need to sense the current flowing through each motor to implement a torque control loop. This is done using a sense resistor, shown in Figure 4.4, followed by an

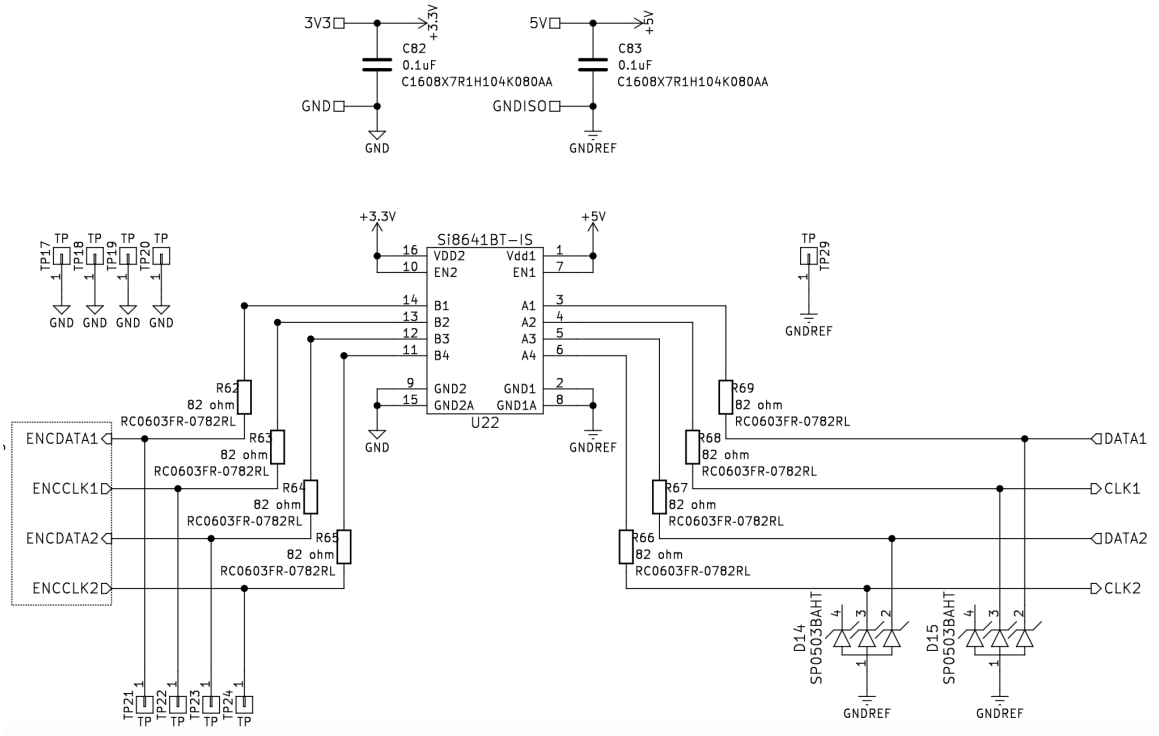


Figure 4.5: Encoder Level Shifting Circuitry

amplifier and some filtering, shown in Figure 4.6. The current flowing through the 1.5mΩ

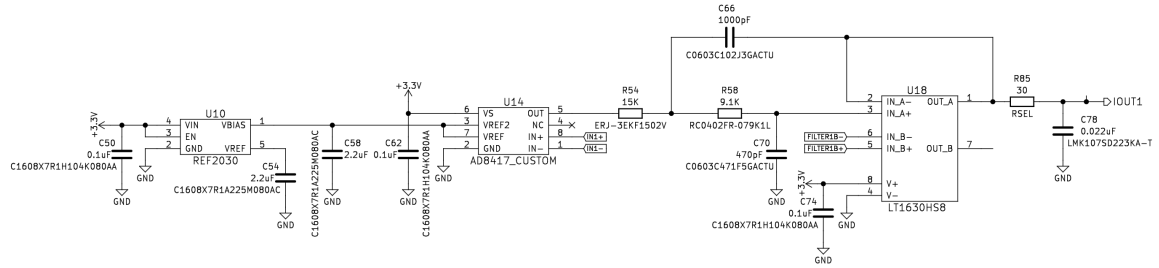


Figure 4.6: Current Sensing Circuitry

sense resistor, R_{Sense}, creates a voltage, V_{Sense}, across the inputs of the AD8417 current sense amplifier. This voltage is expressed by

$$V_{\text{Sense}} = I_{\text{Motor}} R_{\text{Sense}} \tag{4.5}$$

which then creates a voltage at the output of the current sense amplifier, V_{CS}, expressed by

$$V_{\mathrm{CS}} = G_{\mathrm{Amp}} V_{\mathrm{Sense}} + V_{\mathrm{REF}} \tag{4.6}$$

where G_{Amp} is the gain of the AD8417 ($60\ \mathrm{V/V}$ in this case). This causes the current signal, biased about the 1.5V reference signal V_{REF}, to increase or decrease in response to a positive or negative current respectively. This analog signal is passed through a second order Butterworth Sallen-Key low-pass filter to eliminate any high-frequency noise due to transistor switching or the 90 MHz micro-controller clock. This filter has a transfer function of

$$H(s)_{\mathrm{Butterworth}} = \frac{\omega_0^2}{s^2 + \frac{\omega_0}{Q} s + \omega_0^2} \tag{4.7}$$

a cutoff frequency of 20 kHz ($\omega_0 = 124849\ \mathrm{rad/s}$), and $Q = 0.707$. This filter has a gain of $1\ \mathrm{V/V}$ in the pass-band as shown in Figure 4.7. This filtered analog voltage is passed to

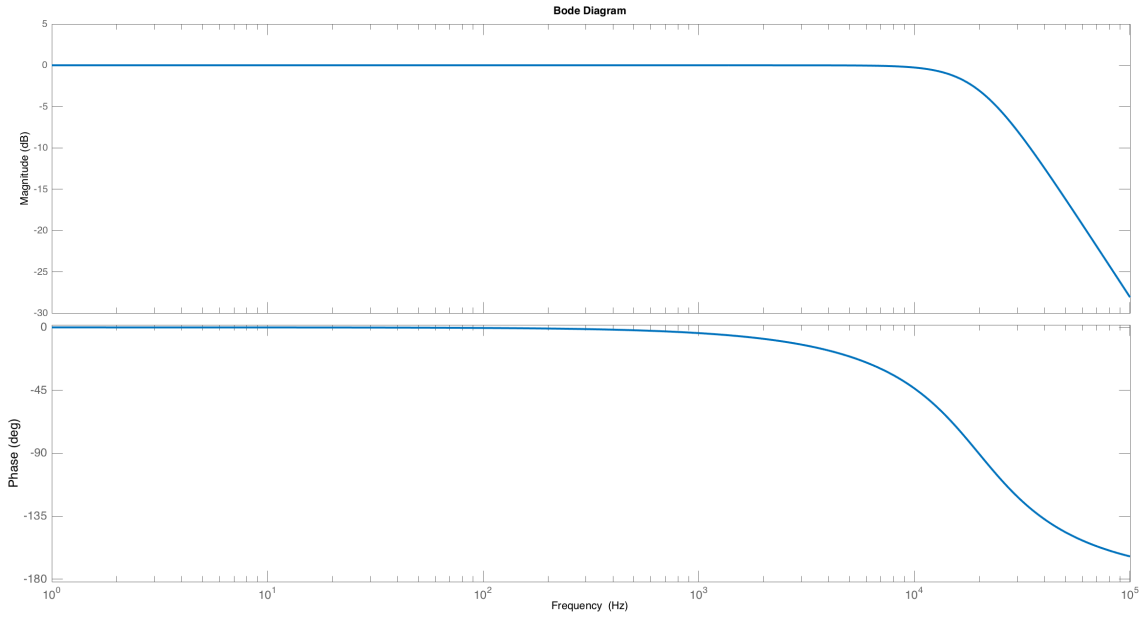


Figure 4.7: Butterworth Filter Frequency Response

the micro-controller's analog-to-digital converter (ADC) and quantized. The quantization

of the signal

$$\Delta_{\mathrm{ADC}} = \frac{V_{\mathrm{Range}}}{2^b} = \frac{3.3V}{2^{12}} \tag{4.8}$$

is determined by the voltage sensing range, V_{Range}, and the number of ADC bits, b. Therefore, the complete end-to-end calculation of ADC integer value created by a motor current can be written as

$$\begin{aligned} N_{ADC} &= \mathrm{ceiling}((\frac{1}{\Delta_{\mathrm{ADC}}})(V_{\mathrm{CS}} + V_{\mathrm{REF}})) \\ &= \mathrm{ceiling}\Big(\Big(\frac{2^{12}}{3.3V}\Big)(I_{\mathrm{Motor}} R_{\mathrm{Sense}} G_{\mathrm{Amp}} + V_{\mathrm{REF}})\Big) \end{aligned} \tag{4.9}$$

Equation 4.9 can be re-arranged as

$$I_{\mathrm{Motor}} = \frac{\Big(N_{\mathrm{ADC}}\Big(\frac{3.3V}{2^{12}}\Big) - V_{\mathrm{REF}}\Big)}{R_{\mathrm{Sense}} G_{\mathrm{Amp}}} \tag{4.10}$$

and used to calculate the motor current based off the integer value stored in the ADC result register and used in code.

The full schematic is included in Appendix A for reference.

4.1.3 HWIL PCB Layout

After the schematic design was finished, we proceeded to implement the design on a printed circuit board (PCB). Great care was taken to minimize the digital noise on analog signals due to the micro-controller clock and the high current switching in the motor drive circuitry. The first way noise was minimized was through ground plane separation. By positioning the motor drive circuity at one edge of the board, we were able to make a cut in the ground plane between the motor control and analog circuitry, taking care not to cut the plane under any traces and inadvertently create return current loops. This cut, shown in Figure 4.8, has the effect of raising the impedance of the motor drive return current path across the cut, forcing the majority of the large motor switching return currents to flow around the edge of

the board, effectively segregating the motor return current path from the analog and digital sections of the board.

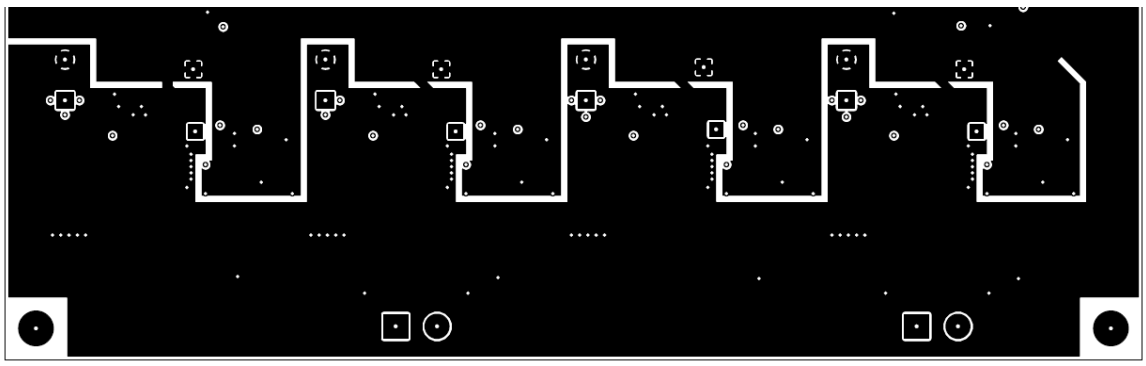


Figure 4.8: Ground Plane Cut

The second way that noise was minimized was through power rail decoupling and bypassing. This was done by placing small value ceramic and larger value tantalum decoupling capacitors close to the power pins of all the integrated circuits (ICs). The small ceramic capacitors serve as a local source of charge and can rapidly release their energy, making them more efficient at eliminating high frequency noise from the power rail. The larger tantalum capacitors are more effective at releasing larger amounts of charge at a slower rate, helping prevent the power rail from being pulled down when circuitry turns on or the motors begin switching, compensating for the parasitic inductance of the power plane. In addition, these capacitors serve as a local AC ground for each IC power pin, minimizing the high frequency return current loop area and allowing us to reliably use the small signal transistor model for the IC [11]. The 3.3V power plane for the micro-controller was further decoupled from the main 3.3V rail through larger filtering capacitors and a ferrite bead. This added impedance to the flow of current in the form of a small inductor, creating a DC short but keeping any high frequency signals from being conducted via the power rail to other circuitry.

Finally, when laying out the analog circuitry, special attention was given to the area of the conductor attached to the inverting node of the amplifier. Our aim was to keep this

trace as short as possible to eliminate noise coupling in through this high impedance node and reduce parasitic capacitance to improve amplifier stability. Switching signals such as PWM outputs and clocks were routed away from this circuitry in an effort to increase the physical separation of the sensitive signals.

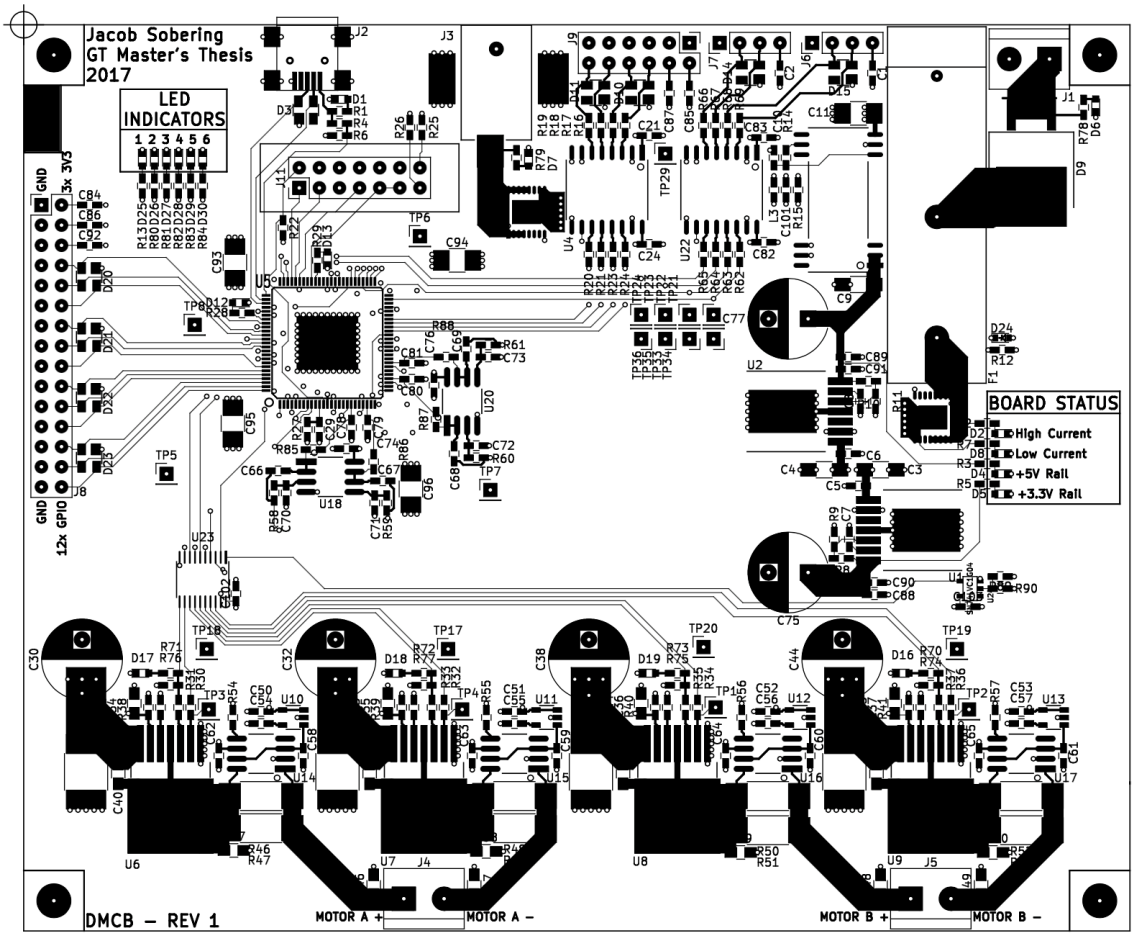


Figure 4.9: HWIL Motor Controller Board Layout (Top Copper and Silkscreen)

Figure 4.9 shows the board layout, while all Gerber files are located in Appendix A.

4.2 Software Design

The final step of the system design was to create the MATLAB® simulation code and the HWIL C code to run on the micro-controller. This section will detail the design process for each coding language from a high level perspective.

4.2.1 MATLAB® Simulation Design

The first step in designing the code was to define a time scale for the simulation. We must treat the physics of the system as continuous time while running the controllers in discrete time. This is done in simulation by running the continuous time portion of the code at an adequately higher rate than the discrete time controllers. To do this we defined a discrete time sample period that would later match the update rate of our micro-controller's ISR, or $T_{\text{sample}} = \frac{1}{30\text{kHz}}$. Next, a continuous time sample period of $T_{\text{cont}} = \frac{T_{\text{sample}}}{100}$ was generated in order to accurately simulate the continuous time nature of the system while still running in a reasonable amount of time.

Next, we implemented the continuous time plant models using their state space models from Equations 2.3, 2.4, and 2.6, shown in Figure 4.10. This section of code was wrapped in a loop that executed every T_{cont} seconds. We wrapped this loop around a second loop that executed every T_{sample} seconds, in which we implemented the control logic defined in Equations 3.2, 3.9, 3.15. Data logging was added, motor parameters were defined, and initial conditions were set. Finally, we determined the tests needed to verify the motor models and characterize the controller. These results are detailed in Chapter 5. The full suite of MATLAB® Code is located in Appendix B.

4.2.2 C Code Design

The next step of the system design was to implement the controllers in hardware. We began by setting up the micro-controller clocks and peripheral systems. The system clock was set to 90 MHz, the timer used for control logic was set to 30kHz, and GPIO, PWM, and ADC registers were set up. For this application, we triggered the ADC sampling off of the PWM signal, syncing the trigger with the center of the positive PWM cycle. This allowed the current waveform coming from the analog sensing circuitry time to settle to a steady state value before being sampled. Within the timer ISR, we implemented the same controllers from Equations 3.2, 3.9, 3.15 as well as code to read the value from the encoder, calculate

```
% plant physics (continuous-time update)
%Drive elec
x_dm = x_dm+h*(A_dme*x_dm+B_dme*Vd+E_dme*omega_dm); %w_dm = sys_omega
T_dm = C_dme*x_dm;

%Load elec
x_lm = x_lm+h*(A_lme*x_lm+B_lme*Vl+E_lme*omega_lm); %w_lm = -sys_omega
T_lm = C_lme*x_lm;

%Geartrain (Forward)
T_sys =  T_dm/Drive.Geartrain.GearRatio - ...
    T_lm/Load.Geartrain.GearRatio*Load.enable;

%mech
x_sys = x_sys+h*(A_sys*x_sys+B_sys*T_sys);
P_sys = C_sys*x_sys;

%Geartrain (Reverse)
omega_dm = x_sys(2)/Drive.Geartrain.GearRatio;
omega_lm = -x_sys(2)/Load.Geartrain.GearRatio;

P_dm = P_sys/Drive.Geartrain.GearRatio;
P_lm = -P_sys/Load.Geartrain.GearRatio;

Pddotdot_sys = (x_sys(2)-store_sys)/T;
store_sys = x_sys(2);
omegadot_ls = -Pddotdot_sys/Load.Geartrain.GearRatio;
```

Figure 4.10: MATLAB® Simulation Plant Models

the motor currents from the sampled ADC value, and output the new motor terminal voltage via PWM. The full suite of C code is located in Appendix C.

CHAPTER 5

RESULTS

With all the controller designs completed, the hardware programmed, and the simulation running, we begin checking the models used in simulation against the physical hardware. This step guarantees the validity of the simulation when generating results. This chapter will detail the model verification tests used and discuss the results of these tests. In addition we will generate the final results of this research and characterize the operating range of the controller.

5.1 Model Verification

Model verification is an important step in conducting research. Without verification, the results of our simulation are suspect. This section will show the tests that were run and their results in order to verify the accuracy of the models used in our simulation.

5.1.1 Stall Current Test (Single Motor, No Control)

The purpose of our first test was to check the stall current of the motors. For this test, the drive shaft of each motor was clamped in place, a constant 12V was applied to the motor terminals, and the motor current was logged. By setting $\dot{I}_{\mathrm{D}} = 0$ and $\omega_{\mathrm{D}} = 0$ in Equation 2.1, we expect the stall current to equal

$$I_{\mathrm{D}} = \frac{V_{\mathrm{D}}}{R_{\mathrm{D}}} \tag{5.1}$$

which can also be obtained by using Ohm's Law. Figure 5.1 shows the results of this test. We can see that the transient response of the motor current tracks very well with the simulation. In addition the steady state values are almost identical. These results confirm

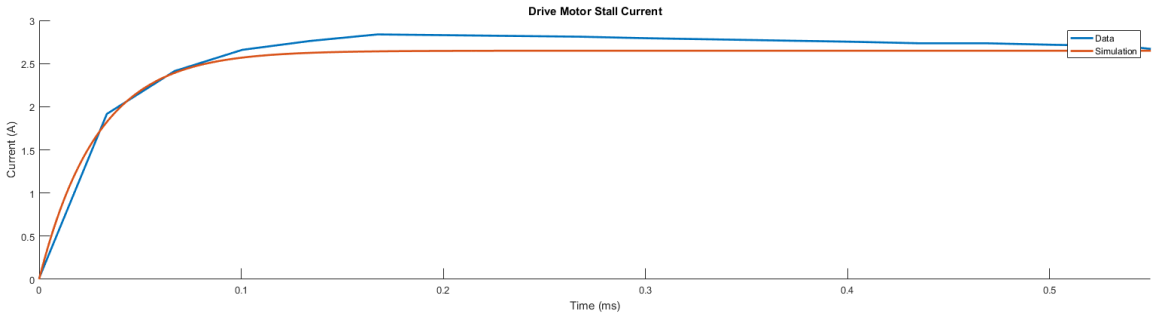


Figure 5.1: Drive Motor Stall Current

the validity of the electrical dynamics of our motor model.

5.1.2 No Load Speed Test (Single Motor, No Control)

The purpose of our second test was to check the no load speed of the motor model. For this test, a constant 12V was applied to the motor, and the position, speed, and current through the motor were logged. This data was then compared to the simulated data generated in MATLAB® under the same conditions. Using Equations 2.1 and 2.5, when a constant 12V is applied we expect the motor to have a steady state speed after the gearbox equal to

$$\omega_{\mathrm{SYS}} = G_{\mathrm{D}} \frac{V_{\mathrm{D}} k_{\mathrm{tD}}}{R_{\mathrm{D}} b_{\mathrm{SYS}} + k_{\mathrm{tD}}^2} \tag{5.2}$$

In addition, we expect to see a steady state current flowing through the motor equal to

$$I_{\mathrm{D}} = \frac{V_{\mathrm{D}}}{R_{\mathrm{D}}} - \frac{V_{\mathrm{D}} k_{\mathrm{tD}}^2}{R_{\mathrm{D}}^2 b_{\mathrm{SYS}} + R_{\mathrm{D}} k_{\mathrm{tD}}^2} \tag{5.3}$$

The results of this test are shown in Figure 5.2. We can see that there is a slight but definite parameter mismatch between the simulated values (taken from the motor datasheet) and the physical motor constants. This is best illustrated by the system speed sub-graph of Figure 5.2 where the physical motor speed is slightly higher than simulated. This could be caused by less friction in the physical motor than expected, a lower or temperature dependent

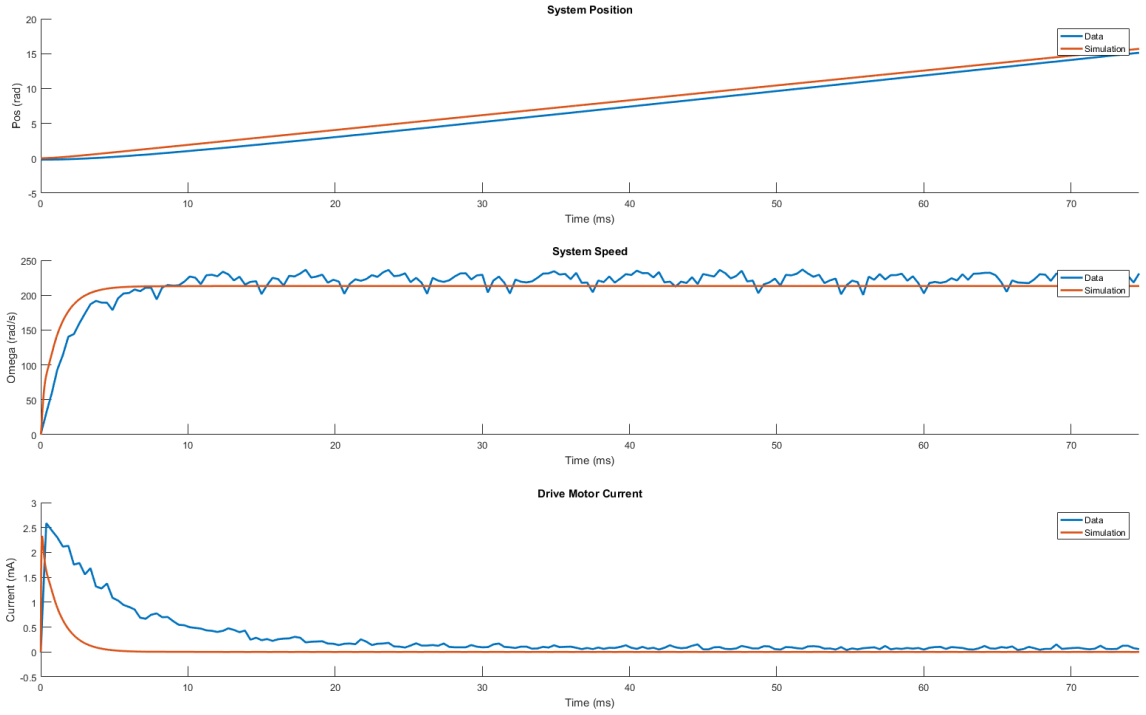


Figure 5.2: Drive Motor No Load Speed Test Results

motor resistance, or a number of other manufacturing variances. These variances are to be expected, and can be lessened by measuring the motor constants and using them in simulation. For this research we chose not to tune the simulation constants in this way, however, and instead wish to see how the parameter mismatch effects the final system. Other than the parameter mismatch, we notice that the speed and current match what we would expect to see from Equations 5.2 and 5.3.

5.1.3 Step Response Test (Single Motor, Position Control)

The purpose of our final test was to verify the controller step response with the simulation. For this test, the motor was started at some unknown initial position. The motor was then commanded to 45 degrees (0.785398 radians) and the position and speed were logged. We expected the controller to settle in approximately $5\,\tau_{\mathrm{DP}} = 0.125$ to $10\,\tau_{\mathrm{DP}} = 0.250$ seconds. Figure 5.3 shows the motor position and speed during the test. We can see that the settling

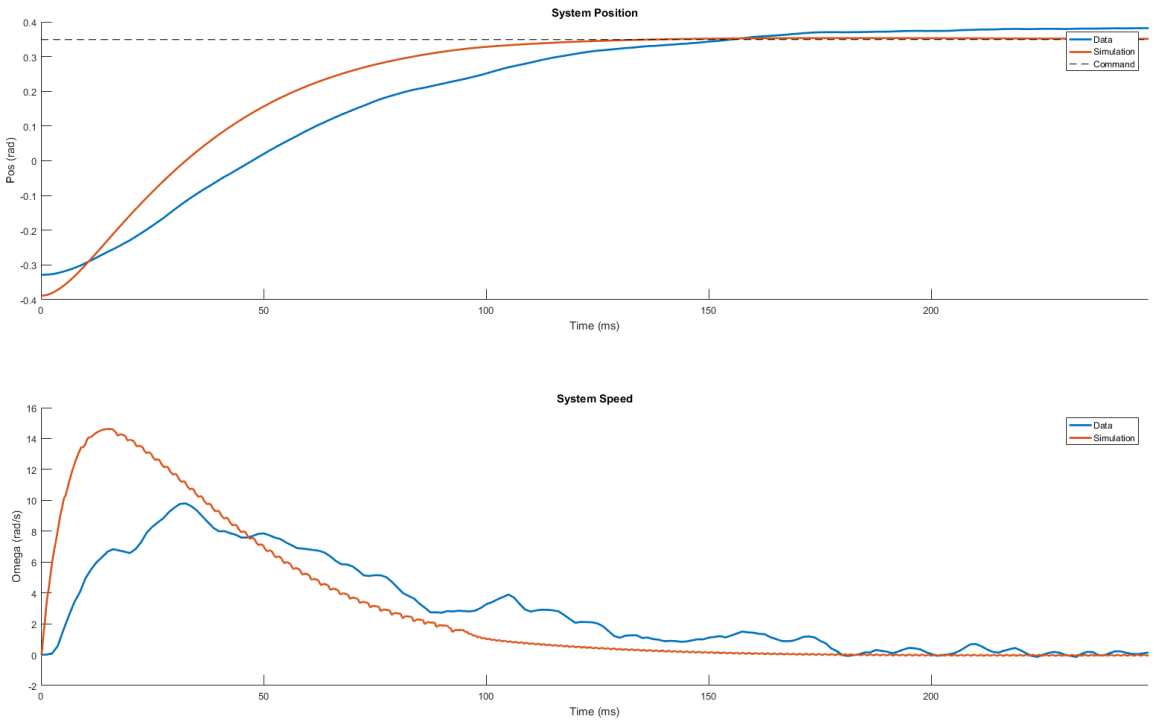


Figure 5.3: Hardware vs Simulation Comparison of Position Controller Step Response

time is approximately 0.175 seconds, or 7 τ_{DP}, which is acceptable for this application. In addition, we see that the HWIL setup velocity estimation has a small amount of noise, but the signal still tracks the simulated motor velocity. This noise can be attributed to the quantization of the encoder and the velocity estimation sampling rate.

5.2 Controller Characterization

The goal of this research was to determine a range of values of D_{CP} for which the loaded position controller still functions. To do this, we implemented the full position controller on the drive motor and the full torque controlled load model on the loading motor as seen in Figure 2.2. We simulated the step response of the loaded position controller over a range of D_{CP} with the rocket parameters defined in Section 2.2.2 and found the settling time and overshoot. Figure 5.4 shows the system position when D_{CP} was varied, first positively and then negatively. The same test was also run on the designed hardware and compared

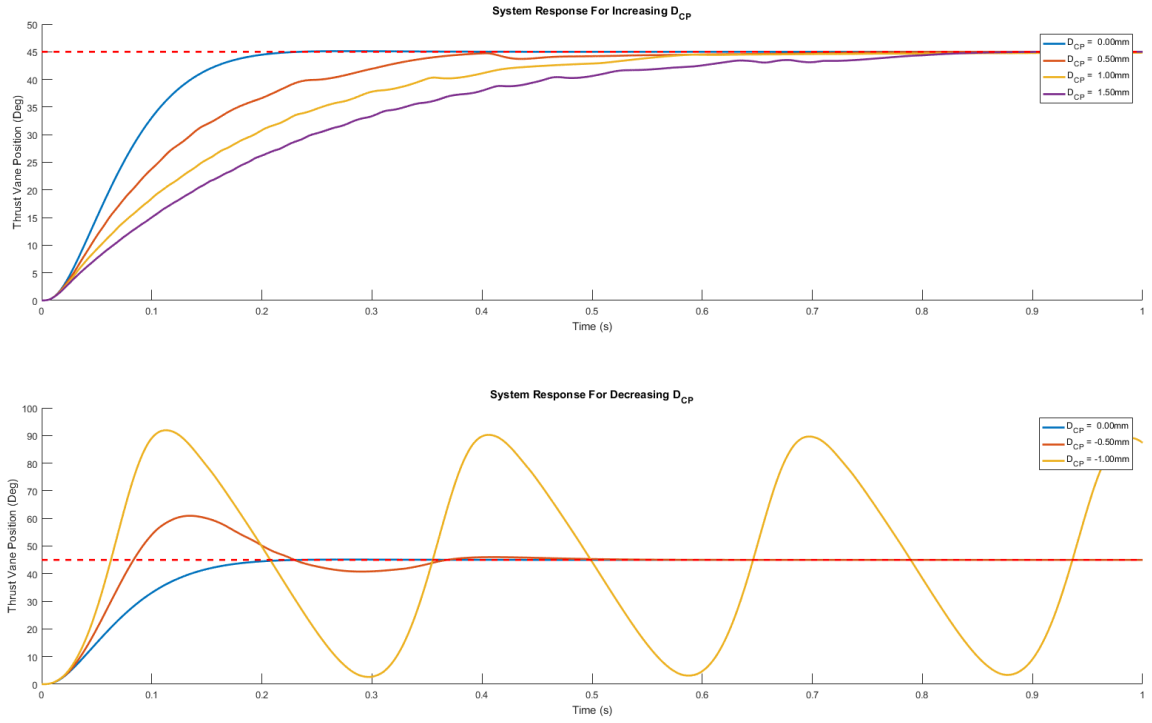


Figure 5.4: Simulation of System Response vs Time with Varying D_{CP}

to the simulation data. The results of this comparison can be seen in Figure 5.5. As expected, there is a slight mismatch between the hardware and simulation data caused by the manufacturing variances discussed previously. However, the results are strikingly similar. When the distance D_{CP} was increased we noticed an increase in the settling time, but no overshoot. This was due to the aerodynamic forces acting in the opposite direction of the controller output, manifesting as a negative disturbance to the control system. This negative disturbance can be overcome by the position controller at steady state. As D_{CP} is increased further, there comes a point at which the maximum thrust vane angle begins to decrease. This happens because, due to Equation 2.11, T_{CP} increases with fin angle and the aerodynamic forces begin to balance out with the maximum torque of the motor. We eventually reach a point where the aerodynamic forces are dominant and the motor cannot overcome them. When D_{CP} is decreased, we notice both increased settling time and overshoot. This is caused by the aerodynamic forces acting in the same direction of the

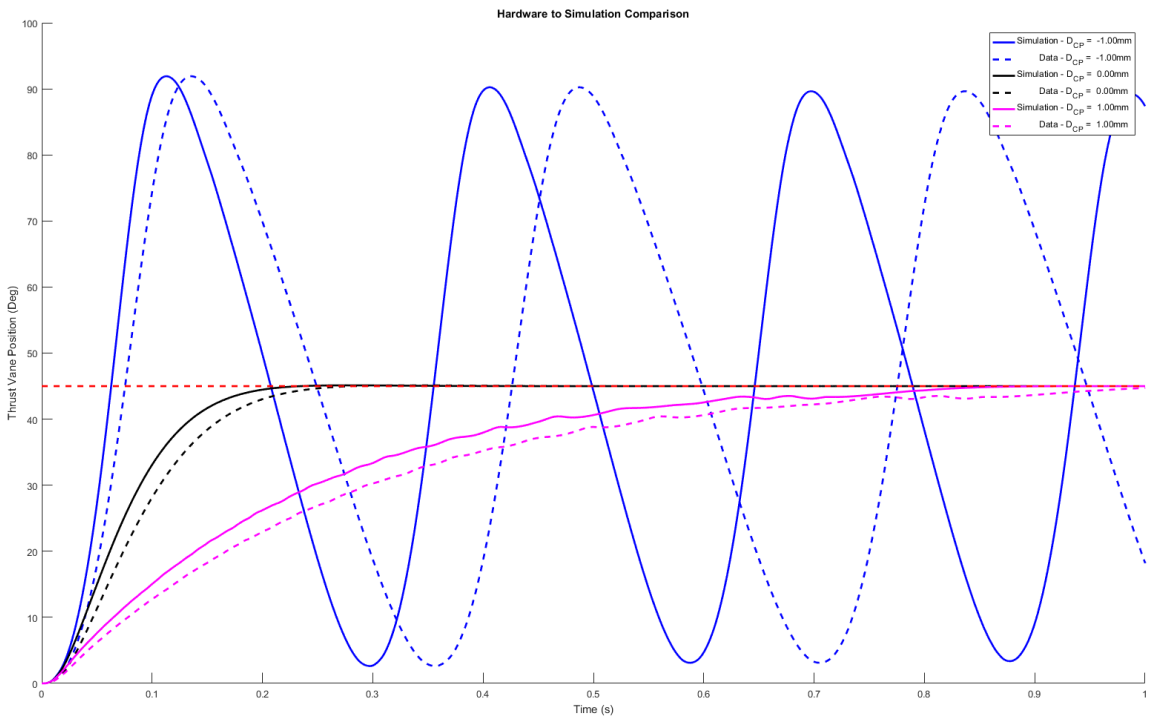


Figure 5.5: Hardware vs Simulation Comparison with Varying D_{CP}

controller output, manifesting as a positive disturbance to the control system. This positive disturbance causes an instability in the control loop and forces overshoot and ringing. Ultimately we reach a point where the controller goes unstable and cannot settle. The settling time and overshoot values found from our test runs are plotted against D_{CP} in Figure 5.6. This figure clearly shows the increase in settling time but not overshoot as D_{CP} increases. In addition it shows how the overshoot increases and the system eventually never settles (becomes unstable) as D_{CP} decreases.

This effect can be mathematically explained using linear approximation of the non-linear load model about the equilibrium point $\theta_{\mathrm{SYS}} = \pi/4$. To do this, we must first take the original mechanical model of the system in Equation 2.5 and substitute in the load

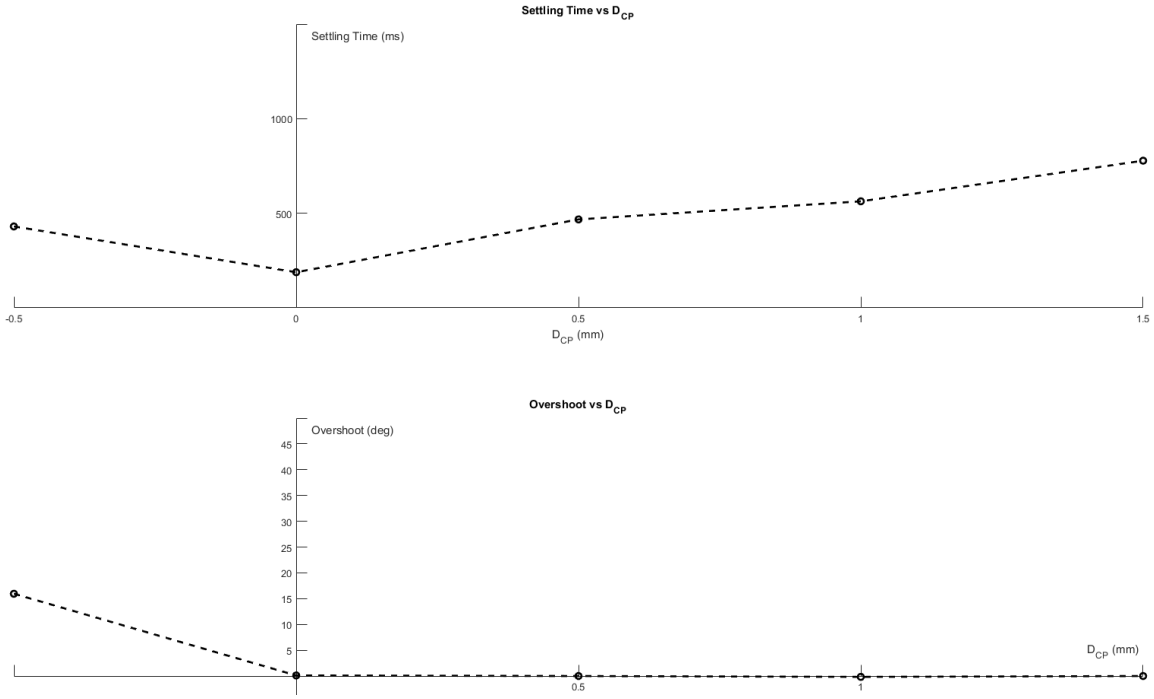


Figure 5.6: Simulation of Settling Time and Overshoot vs D_{CP}

model from Equation 2.14. This gives us the non-linear model for the system:

$$J_{\mathrm{SYS}}\dot{\omega}_{\mathrm{SYS}} = T_{\mathrm{D}} - (m_{\mathrm{V}}\|a_{\mathrm{R}}\|\, D_{\mathrm{CG}}\cos(\theta_{\mathrm{SYS}}) + \frac{A_{\mathrm{V}}}{A_{\mathrm{RM}}}\|F_{\mathrm{RM}}\|\, D_{\mathrm{CP}}\cos(\theta_{\mathrm{SYS}}) + J_{\mathrm{V}}\dot{\omega}_{\mathrm{SYS}}) - b_{\mathrm{SYS}}\omega_{\mathrm{SYS}} \quad (5.4)$$

Since we have found that D_{CP} is the dominant parameter associated with position controller stability, we can define the following constants

$$C_0 = m_{\mathrm{V}}\|a_{\mathrm{R}}\|\, D_{\mathrm{CG}} \quad (5.5a)$$

$$C_1 = \frac{A_{\mathrm{V}}}{A_{\mathrm{RM}}}\|F_{\mathrm{RM}}\| \quad (5.5b)$$

in order to simplify Equation 5.4. Adding state space integral control yields the following

three equations:

$$\dot{\theta}_{\mathrm{SYS}} = \omega_{\mathrm{SYS}} \tag{5.6a}$$

$$\dot{\omega}_{\mathrm{SYS}} = \frac{-K_{11\mathrm{DP}}\theta_{\mathrm{SYS}} - (K_{12\mathrm{DP}} + b_{\mathrm{SYS}})\omega_{\mathrm{SYS}} - K_{2\mathrm{DP}}\sigma_{\mathrm{SYS}} - (C_0 + C_1 D_{\mathrm{CP}})\cos(\theta_{\mathrm{SYS}})}{J_{\mathrm{SYS}} + J_{\mathrm{V}}} \tag{5.6b}$$

$$\dot{\sigma}_{\mathrm{SYS}} = \theta_{\mathrm{SYS}} - r_{\mathrm{SYS}} \tag{5.6c}$$

Since this model of the system is non-linear it cannot be expressed in the traditional state space form, but instead can be represented simply in the form

$$\dot{x} = f(x) \tag{5.7}$$

where x is the array of state space variables or the column vector of system angle, θ_{SYS}, angular velocity, ω_{SYS}, and integral error, σ_{SYS}, or

$$x = \begin{bmatrix} \theta_{\mathrm{SYS}} \\ \omega_{\mathrm{SYS}} \\ \sigma_{\mathrm{SYS}} \end{bmatrix}$$

By definition, there is no motion at the equilibrium point and therefore

$$0 = f(\bar{x}) \tag{5.8}$$

We can then use linear approximation to define

$$\bar{A} = \left.\frac{\partial f}{\partial x}\right|_{(\bar{x})} \tag{5.9}$$

and can now represent our approximated system in state space form as

$$\dot{\tilde{x}} \approx \bar{A}\tilde{x} \tag{5.10}$$

where $\tilde{x}$ is the difference between the real state space variables and the equilibrium state space variables, or

$$\tilde{x} = x - \bar{x} \tag{5.11}$$

Using Equation 5.9 on the system of equations in Equation 5.6, we can find our approximated linear matrix

$$\bar{A} = \begin{bmatrix} 0 & 1 & 0 \\ -\dfrac{K_{11\mathrm{DP}} - (C_0 + C_1 D_{\mathrm{CP}})\sin(\pi/4)}{J_{\mathrm{SYS}} + J_{\mathrm{V}}} & -\dfrac{K_{12\mathrm{DP}} + b_{\mathrm{SYS}}}{J_{\mathrm{SYS}} + J_{\mathrm{V}}} & -\dfrac{K_{2\mathrm{DP}}}{J_{\mathrm{SYS}} + J_{\mathrm{V}}} \\ 1 & 0 & 0 \end{bmatrix} \tag{5.12}$$

We can clearly see that this matrix differs from the modified $\widehat{A}_{\mathrm{DP}}$ matrix found in Equation 3.3 most importantly with the addition of a term in the middle left value relating to the load. This term causes the eigenvalues of $\bar{A}$ to shift as D_{CP} is changed. Figure 5.7 shows how the eigenvalues move farther into the left half plane ($\mathrm{Re}\{\lambda\} < 0$) as D_{CP} is increased and move into the right half plane ($\mathrm{Re}\{\lambda\} > 0$) as D_{CP} is decreased. Using Lyapunov's Indirect Method we know that our equilibrium point is exponentially stable if and only if every eigenvalue, λ, of $\bar{A}$ satisfies $\mathrm{Re}\{\lambda\} < 0$. Therefore, we can see that as D_{CP} decreases, the two of the eigenvalues of $\bar{A}$ move out of the realm of exponential stability and we begin to see overshoot and eventually instability as discussed above and shown in Figure 5.3. When tested over the interval $-1.5\ \mathrm{mm} < \mathrm{D_{CP}} < 1.5\ \mathrm{mm}$, we found from our eigenvalue analysis that the position control is in the stable region if $D_{\mathrm{CP}} > -0.006\ \mathrm{mm}$. This matches well with what we saw in the time-domain simulation results, with the position controller being stable for a small range of negative D_{CP} and a large range of positive D_{CP} values.

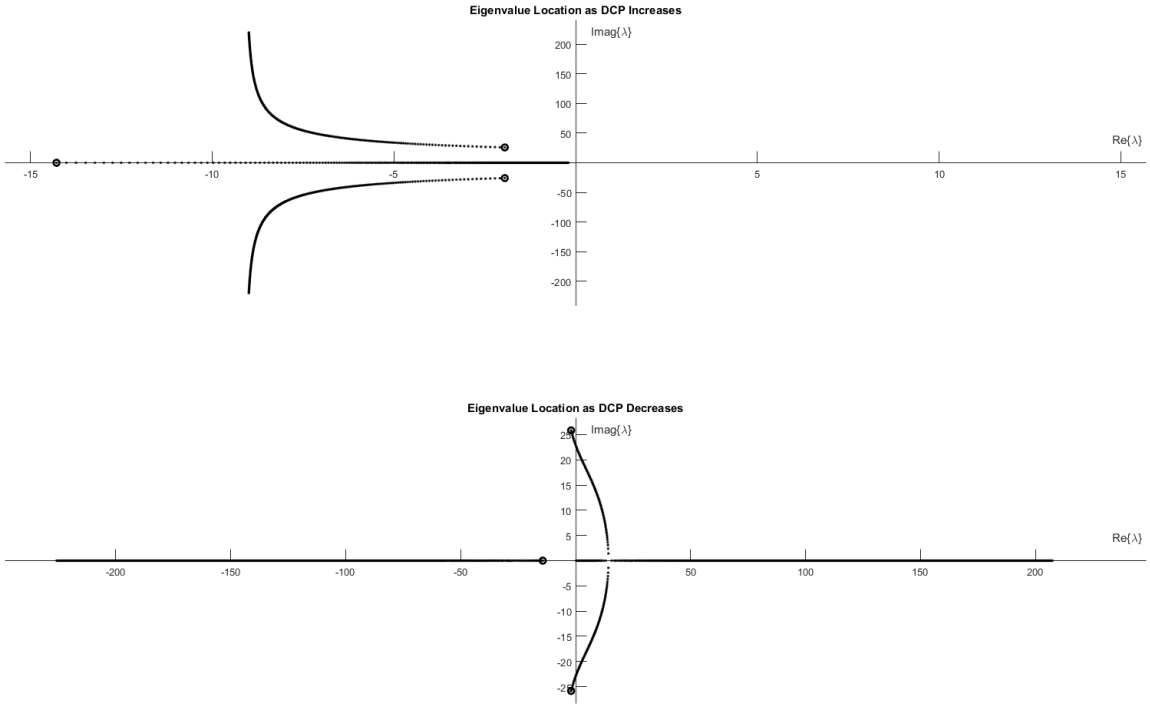


Figure 5.7: Eigenvalue Locations of $\bar{A}$ with Varying D_{CP}

CHAPTER 6
CONCLUSION

Control of missiles and rockets is very complex and vane degradation adds a unique twist to the research being conducted. This research was based around the desire to control the angular position of a small thrust vane that was known to degrade during flight. Since the degradation boundaries were not defined, we decided instead to provide the range of values for which the position controller still functioned adequately. To do this we needed background knowledge of TVC and, more specifically, how exhaust flow deflection TVC works. The literature study gave us the working knowledge we needed and allowed us to begin our physical modeling of the system.

Modeling gave us a fundamental understanding of the physics of our system, both of the DC motors and the thrust vane. We know that the load on the vane is created by three torques acting together and defined these torques from the parameters given. We then decided on a parameter that was most important when testing the range of vane degradation. It was found that due to the high expected motor exhaust force, the distance from the center of rotation to the center of pressure was the critical parameter. We hypothesized that the effects on the controller would vary greatly depending on the direction in which the center of pressure moved.

When designing the controllers, we treated the vane degradation and load model as a disturbance in order to simplify our system. Due to this, we were able to create integral state space controllers for both position and torque without considering the effects of the load on the position controller. The gains for these controllers were tuned using a single tuning variable which was directly related to the settling time of the loop. In addition, we designed a digital differentiator which can function over the range of values needed in this application. With these key designs completed, we were able to move into hardware

design.

The hardware served to validate the plant models and give us experience in board level design, debugging, and testing. Many components were designed including analog filtering and sensing, motor power stages, micro-controller circuitry, and encoder level-shifting electronics. Once the hardware was complete, both the simulation and HWIL software were generated. With the controller designed and plants modeled, the software design was straightforward. However, the hardware proved invaluable when debugging the software. By knowing the response of the physical DC motors, we were able to fix many errors in the code and validate our models.

This model verification step served to provide confidence in our final results generated in simulation. We were able to note any parameter mismatch between our hardware and simulation and determine if it needed to be addressed. We only saw a slight mismatch, which is likely to happen due to the varying characteristics of any physical motor. Since it is unrealistic to test every motor and tune the controller with these updated values during production, we instead left the small mismatch in the system when generating our final results.

With all hardware and software completed, the models verified, and the testing parameter chosen, we were able to test our controller's response over a defined range of values. We had hypothesized that the load will have a drastically different effect on the controller based off of how the center of pressure drifts and we found this to be true. When the center of pressure drifts up towards the rocket nozzle, or in the negative direction, we see greatly increased overshoot, ringing, and eventually instability. We believe this is caused by the positive feedback effect this introduced force has on the controller. In contrast, when the center of pressure moves away from the nozzle, in the positive direction, we see increased settling time but no instability or overshoot. This is caused by the introduced force acting against the controller with a negative feedback effect. We see that eventually, if the center of pressure moves far enough in a positive direction, the motor will no longer be able to

produce the torque required to turn the vane, as the force from the rocket engine will dominate. Lyapunov stability analysis validated both our hypothesis and our results by showing how the eigenvalues of the controlled system move due to the effects of the load, since these effects were treated as disturbances when designing the controllers.

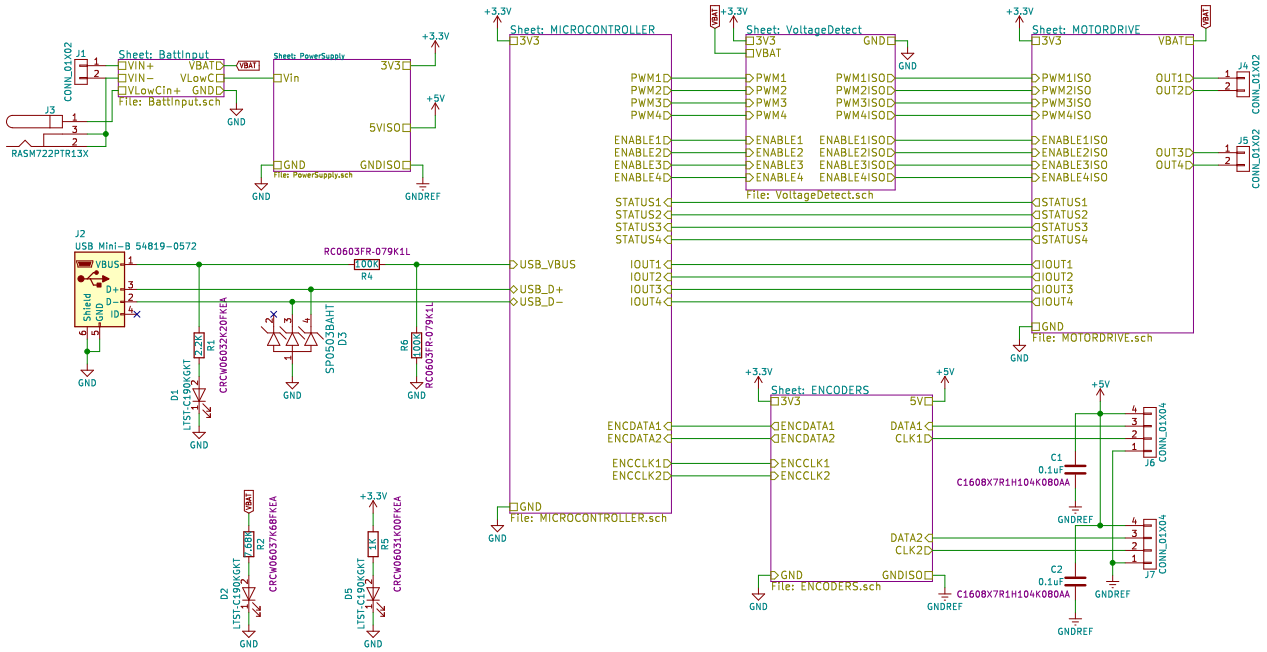
With this research, we designed a method of testing our position controller over a range of degradation values for our thrust vane with a theoretical rocket motor. To further improve this research, we need to simulate using actual rocket engine and thrust vane data. In addition, extensive effort needs to be invested to determine exactly how the thrust vane degrades, as we have found that the direction in which the center of pressure moves plays a critical role in controller stability. Once a model is generated that shows the vane degradation as a function of time, vane angle, and other parameters, a full HWIL test of the entire flight path can be generated and Monte Carlo analysis can be run. That being said, this research provides an excellent starting point and prototype for actual flight hardware and testing and provides us insight into the loading effects of the degrading vane.

Appendices

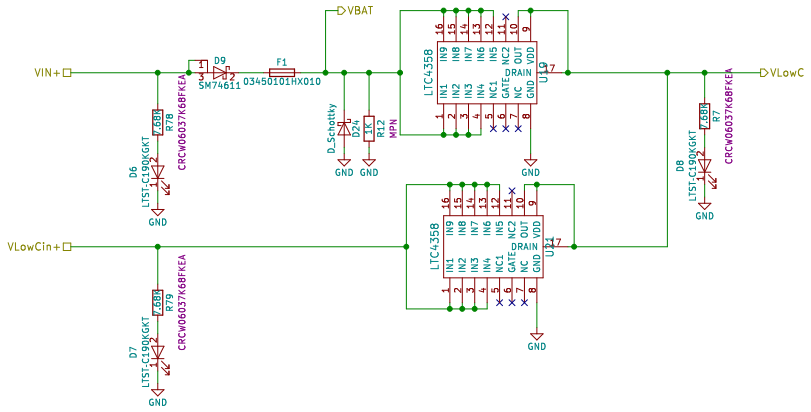
APPENDIX A

HARDWARE DESIGN FILES

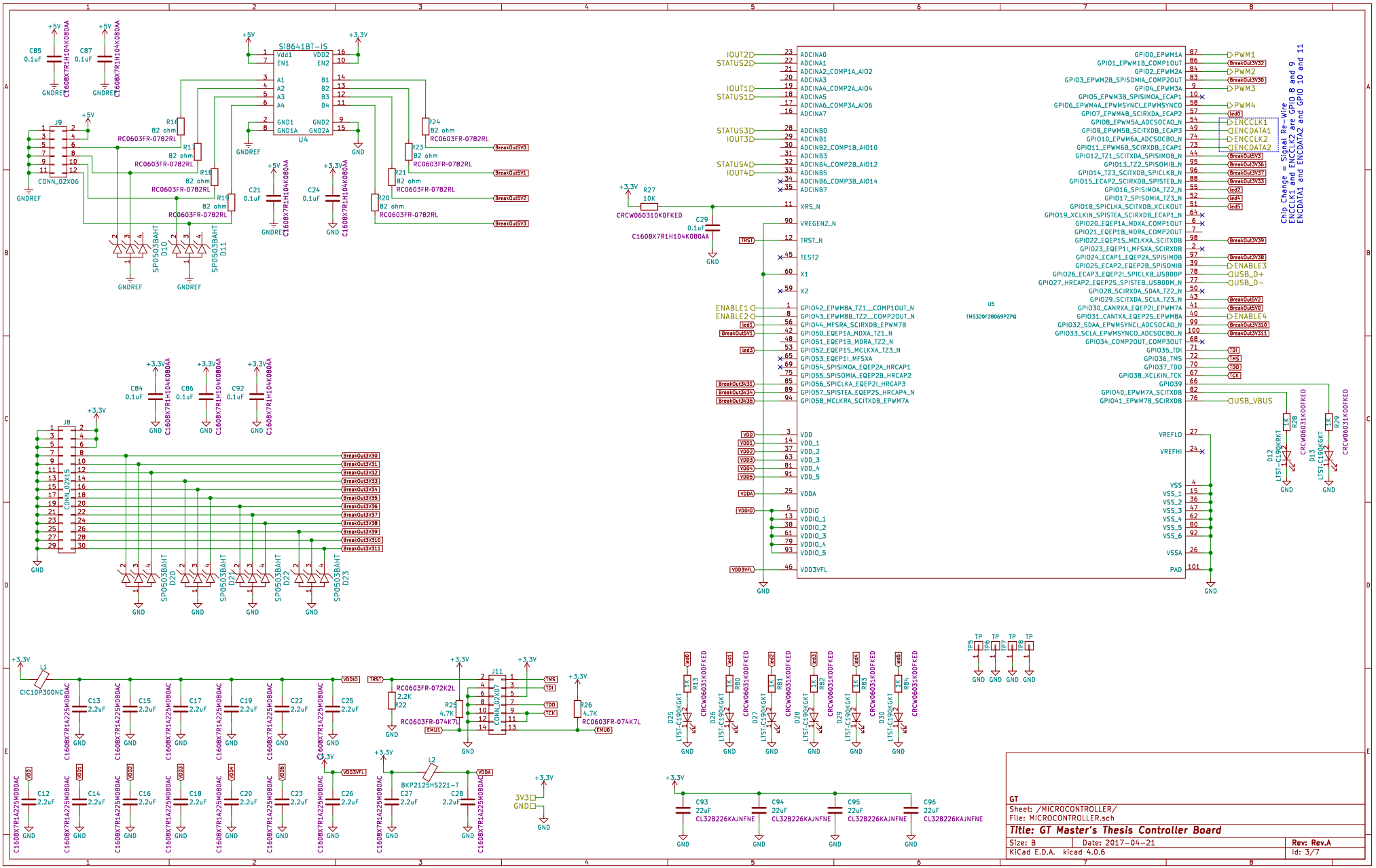
Appendix A contains all hardware related design files that were created for the purpose of this research. All files were generated using KiCad EDA and were created by the author.

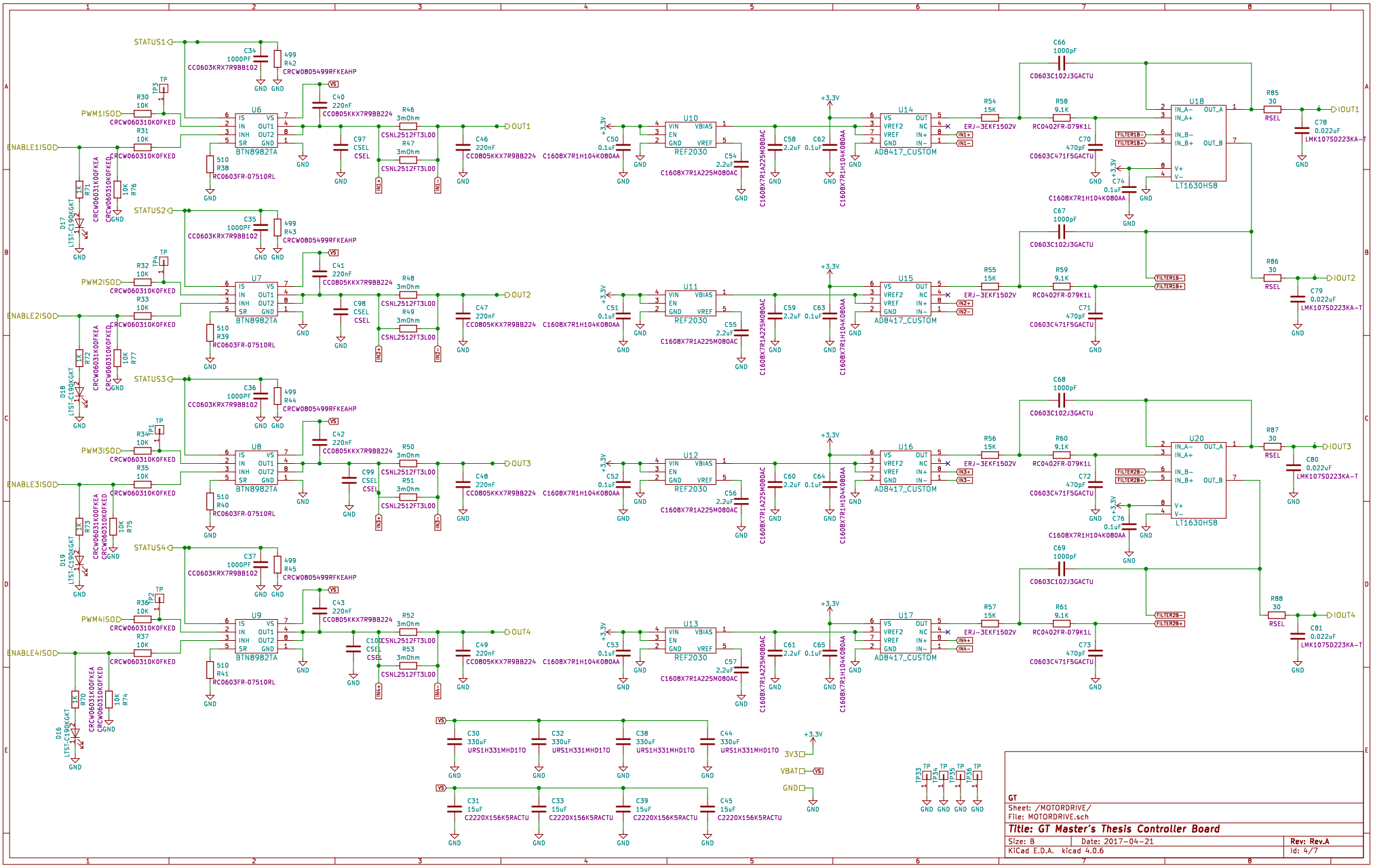


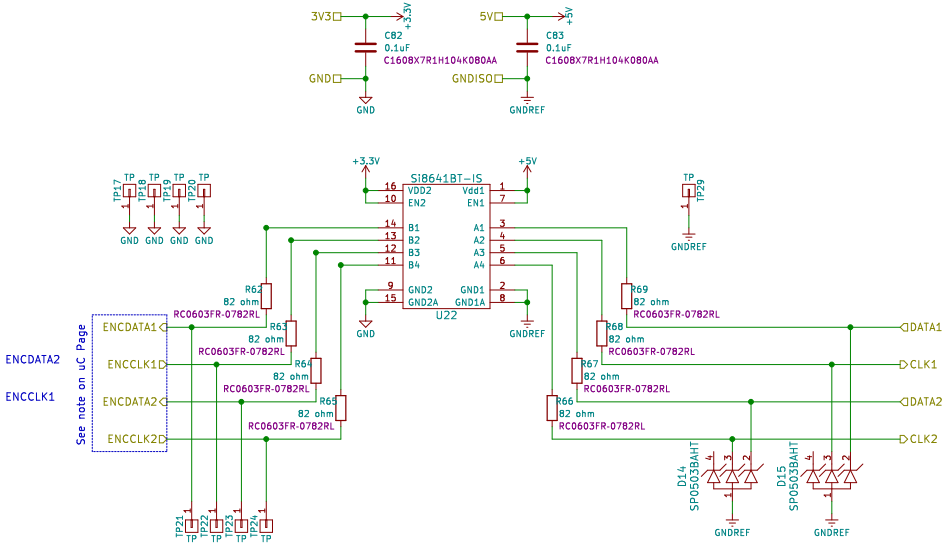
Hierarchy		
GT		
Sheet: /		
File: Controller_Board.sch		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/7



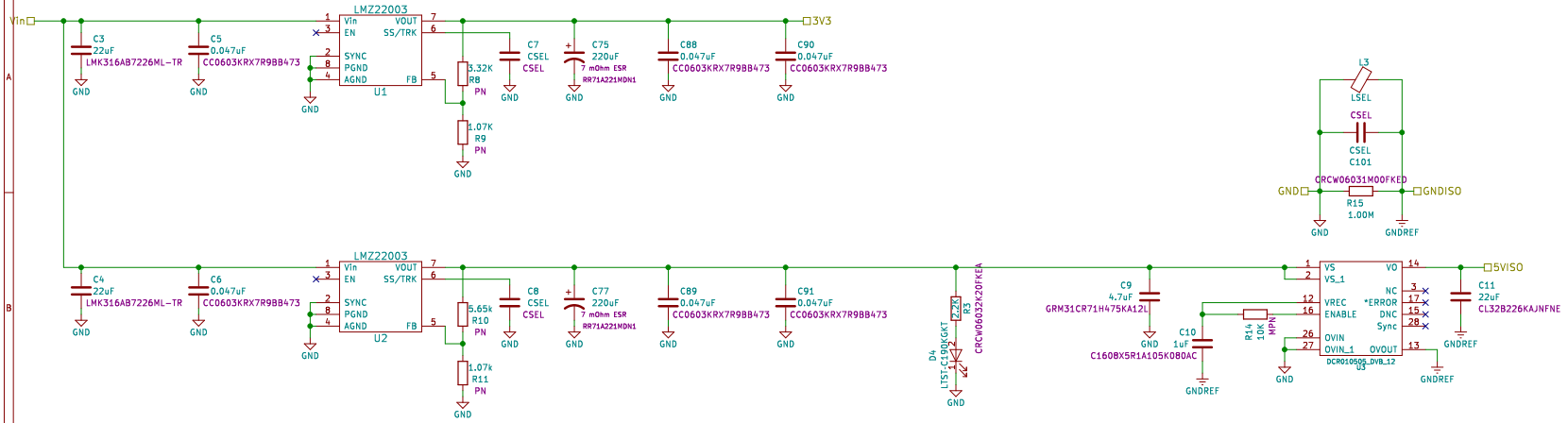
GT		
Sheet: /BattInput/		
File: BattInput.sch		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 2/7


Chip Change = Signal Re-Wire
ENCCLK1 and ENCCLK2 are GPIO 8 and 9
ENCDATA1 and ENCDATA2 and GPIO 10 and 11
TMS320F28069PZPQ
GT
Sheet: /MICROCONTROLLER/
File: MICROCONTROLLER.sch
Title: GT Master's Thesis Controller Board
Size: B
Date: 2017-04-21
KiCad E.D.A. kicad 4.0.6
Rev: Rev.A
Id: 3/7


STATUS1
STATUS2
STATUS3
STATUS4
PWM1ISO
PWM2ISO
PWM3ISO
PWM4ISO
ENABLE1ISO
ENABLE2ISO
ENABLE3ISO
ENABLE4ISO
BTN8982TA
REF2030
AD8417_CUSTOM
LT1630HS8
ERJ-3EKF1502V
OUT1
OUT2
OUT3
OUT4
IOUT1
IOUT2
IOUT3
IOUT4
3V3
VBAT
GND
GT
Sheet: /MOTORDRIVE/
File: MOTORDRIVE.sch
Title: GT Master's Thesis Controller Board
Size: B
Date: 2017-04-21
Rev: Rev.A
KiCad E.D.A. kicad 4.0.6
Id: 4/7



GT		
Sheet: /ENCODERS/		
File: ENCODERS.sch		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 5/7

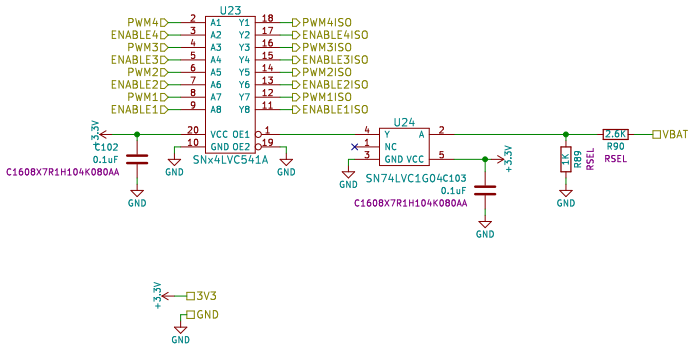


GT

Sheet: /PowerSupply/
File: PowerSupply.sch

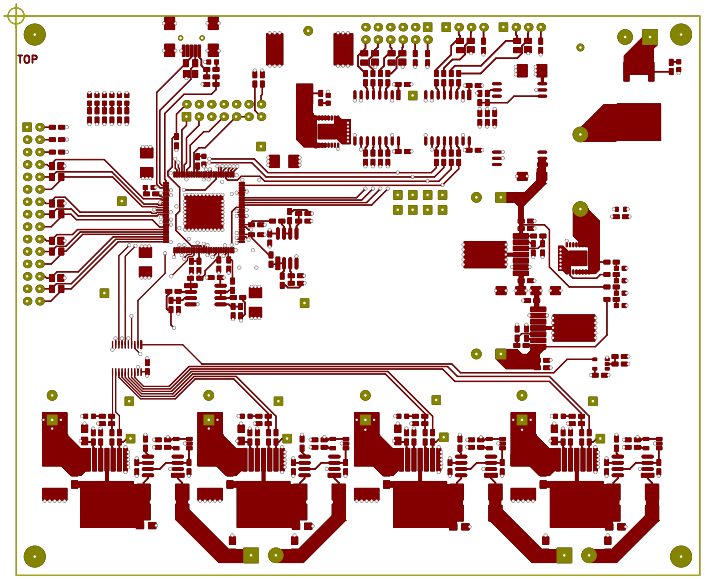
Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 6/7



GT		
Sheet: /VoltageDetect/		
File: VoltageDetect.sch		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 7/7

Top Copper



GT

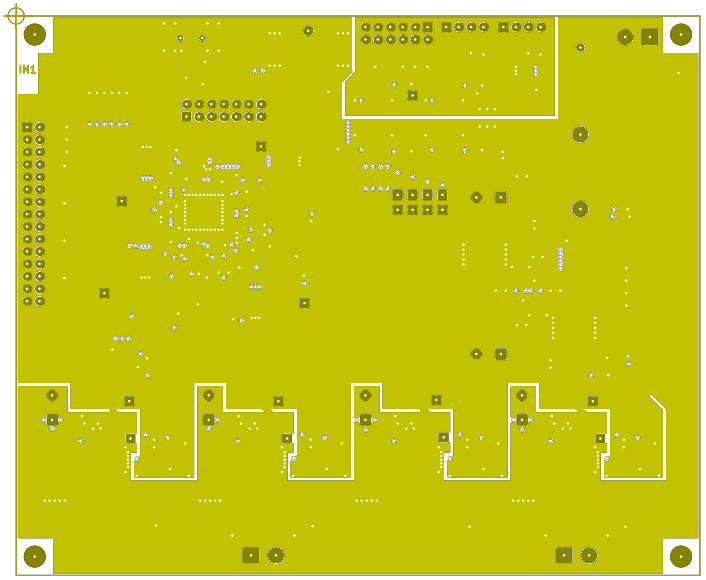
Sheet:

File: Controller_Board.kicad_pcb

Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1

Inner 1

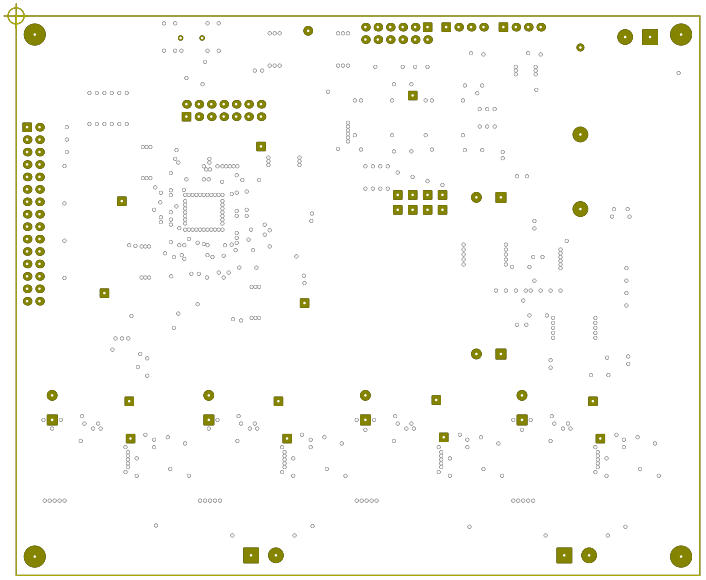


GT

Sheet:
File: Controller_Board.kicad_pcb

Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1



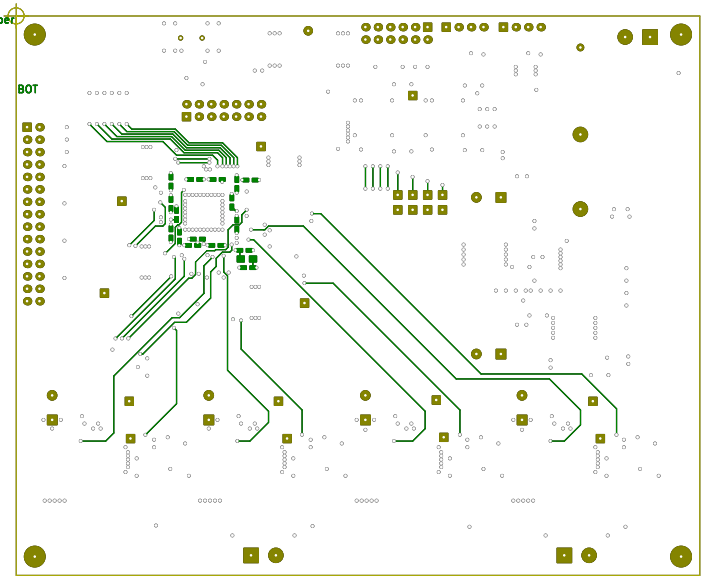
GT

Sheet:
File: Controller_Board.kicad_pcb

Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1

Bottom Copper



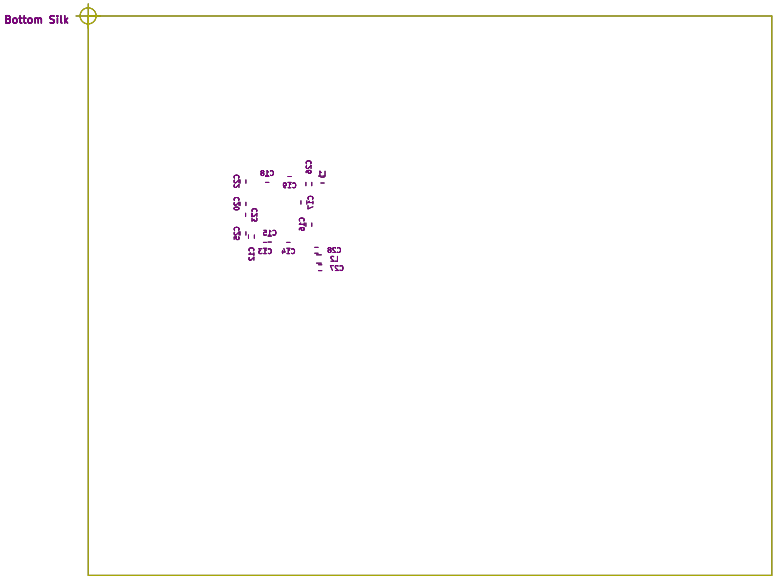
GT

Sheet:

File: Controller_Board.kicad_pcb

Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1



GT

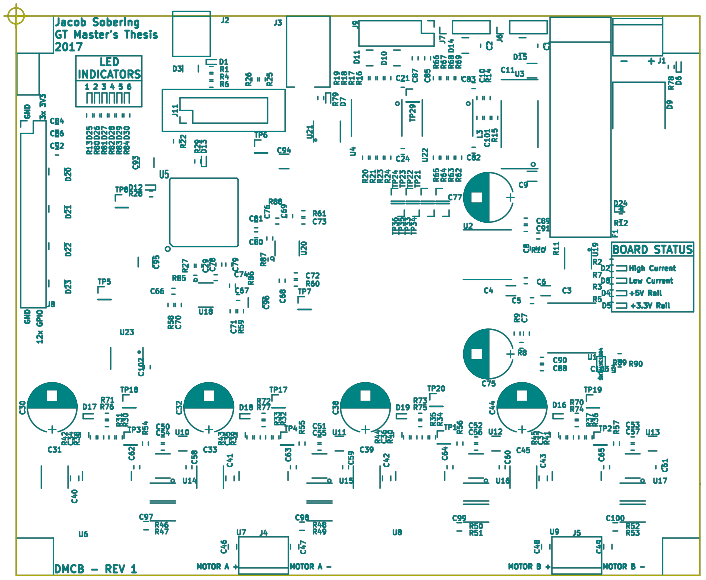
Sheet:

File: Controller_Board.kicad_pcb

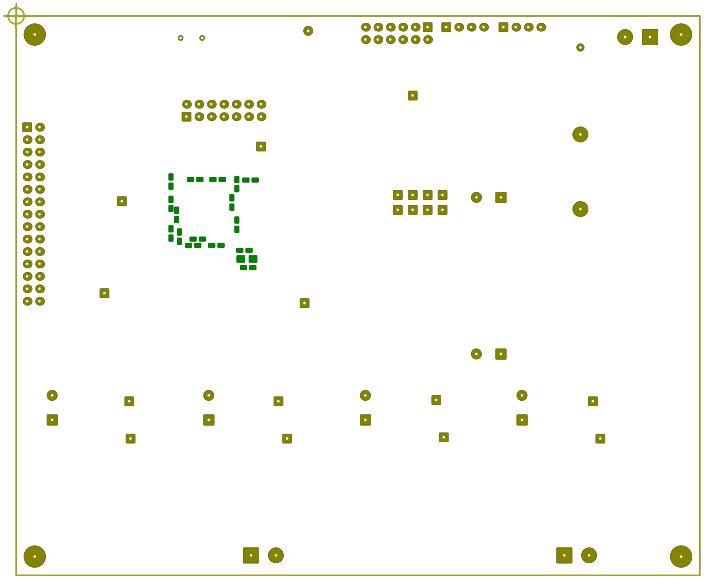
Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1

Top Silk

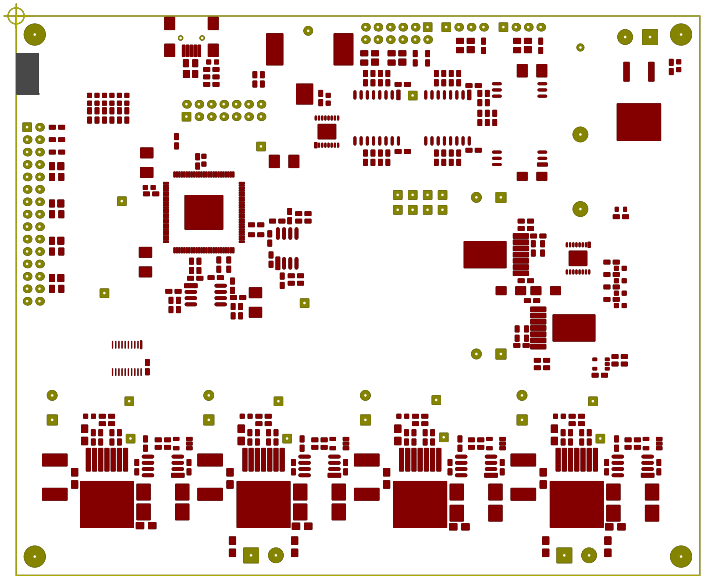


GT		
Sheet:		
File: Controller_Board.kicad_pcb		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1



GT		
Sheet:		
File: Controller_Board.kicad_pcb		
Title: GT Master's Thesis Controller Board		
Size: B	Date: 2017-04-21	**Rev: Rev.A**
KiCad E.D.A. kicad 4.0.6		Id: 1/1

Top Mask



GT

Sheet:

File: Controller_Board.kicad_pcb

Title: GT Master's Thesis Controller Board

Size: B	Date: 2017-04-21	Rev: Rev.A
KiCad E.D.A. kicad 4.0.6		Id: 1/1

APPENDIX B

MATLAB® CODE

B.1 Test_System.m

```
%% Test System Matlab
%clear; clc; close all;

%%
%CDATA_t = xlsread('t.csv');
%CDATA_SYS_P = xlsread('SYS_P.csv');
%CDATA_SYS_P = xlsread('SYS_P_Test.csv');
%CDATA_I_dm = xlsread('I_dm.csv');
%CDATA_I_lm = xlsread('I_lm.csv');
%% plant simulation model parameters
Drive.enable = 1;
Load.enable = 1;

% Drive Motor Constants
Drive.Motor.Vt0 = 12;              %V
Drive.Motor.I0 = 35.4e-3;          %A
Drive.Motor.w0 = 13200*pi/30;      %rad/s

Drive.Motor.R    = 4.53;           %Ohm (4.4311)
%Drive.Motor.R    = 5.15;          %Ohm
Drive.Motor.L    = 0.131e-3;       %H
Drive.Motor.J    = 1.08e-7;        %kg*m^2 %was g*cm^2
Drive.Motor.Kt   = 8.53e-3;        %N*m/A  %was mN*m
Drive.Motor.Kw   = 8.53e-3;        %V*s/rad
Drive.Motor.b    = Drive.Motor.Kt*Drive.Motor.I0/Drive.Motor.w0; %mN*m*s/
    rad

Drive.Geartrain.GearRatio = 1/6.6;

% Load Motor Constants
Load.Motor.Vt0 = 12;               %V
Load.Motor.I0 = 41.6e-3;           %A
Load.Motor.w0 = 13200*pi/30;       %rad/s

Load.Motor.R    = 1.63;            %Ohm
%Load.Motor.R    = 8.694;          %Ohm
Load.Motor.L    = 0.096e-3;        %H  1
Load.Motor.J    = 2.36e-7;         %kg*m^2 %was g*m^2
Load.Motor.Kt   = 8.59e-3;         %N*m/A %%was mN*m
Load.Motor.Kw   = 8.59e-3;         %V*s/rad
Load.Motor.b    = Load.Motor.Kt*Load.Motor.I0/Load.Motor.w0; %N*m*s/rad

```

```
Load.Geartrain.GearRatio = 1/3.9;

% System Constants
%System.Max_Position = deg2rad(50);
System.Max_Position = inf;
%System.Max_Torque = (0.95)*0.0356;
%System.Max_Voltage = (0.95)*12;
System.Max_Voltage = 12;                                    %CHANGE

System.Initial_Position = deg2rad(0);
%System.Initial_Position = deg2rad(50);
%System.Initial_Position = CDATA_SYS_P(1);
System.Initial_Speed = 0;

System.b = Drive.Motor.b*Drive.enable + Load.Motor.b*Load.enable;
System.J = Drive.Motor.J*Drive.enable + Load.Motor.J*Load.enable;

%Load Model
% LE (Leading edge)                                  at 0 in
% C (Chord)                                          at 1.8 in
% Cp (Center of Pressure)                            at 0.39*C in = 0.702in
% R (Rotationa Axis)                                 at Cp
% Cg (Center of Gravity)                             at 0.5*C in = 0.9in
% dcg (distance from rotation axis to CG)            at 0.198 in
% dcp (distance from rotation axis to CP)            at 0in;
% F (Rocket Engine Force)                            at CP (force curve to
    come)

Load.Model.J = 2.5e-6;  %kg*m^2
Load.Model.a = 15;     %m/s^2
Load.Model.m = 0.1;    %kg
Load.Model.dcg = 5e-3; %0.198in to m;
Load.Model.dcg = 0;
%Load.Model.dcp = 0;
Load.Model.F = 9000*(250)/(2*pi*152.4*152.4);

%% controller design parameters
% Drive Position Controller
Controller.Frequency = 30000; % Hz
Controller.Tau = 0.025;
%Controller.Tau = 0.2;
Controller.Lambda = 1/Controller.Tau;
Controller.J = System.J;
Controller.K11 = Drive.Geartrain.GearRatio*Controller.J*3*(Controller.
    Lambda^2);
Controller.K12 = Drive.Geartrain.GearRatio*Controller.J*3*Controller.
    Lambda;
Controller.K2 = Drive.Geartrain.GearRatio*Controller.J*(Controller.
    Lambda^3);

%Drive Current Controller
Drive.PCE.Kt              = 17.8e-3; %Nm/A
%Drive.PCE.Tau             = 0.1e-3; %seconds
Drive.PCE.Tau             = 0.000175; %seconds
```

```
92  Drive.PCE.Lambda          = 1/Drive.PCE.Tau;
93  Drive.PCE.K1              = (2*Drive.Motor.L*Drive.PCE.Lambda - Drive.
        Motor.R)/Drive.Motor.Kt;
94  Drive.PCE.K2              = (Drive.PCE.Lambda^2)*(Drive.Motor.L/Drive.
        Motor.Kt);
95
96  %Load Current Controller Constants
97  Load.PCE.Kt               = 17.8e-3; %Nm/A
98  %Load.PCE.Tau              = 0.1e-3; %seconds
99  Load.PCE.Tau              = 0.000175; %seconds
100 Load.PCE.Lambda           = 1/Load.PCE.Tau;
101 Load.PCE.K1               = (2*Load.Motor.L*Load.PCE.Lambda - Load.Motor.R)
        /Load.Motor.Kt;
102 Load.PCE.K2               = (Load.PCE.Lambda^2)*(Load.Motor.L/Load.Motor.Kt
        );
103
104
105 %% simulation model coefficient matrices
106 A_dme = -Drive.Motor.R/Drive.Motor.L;       %A matrix of drive motor
        electrical
107 B_dme = 1/Drive.Motor.L;                    %B matrix of drive motor
        electrical
108 E_dme = -Drive.Motor.Kw/Drive.Motor.L;      %E matrix of drive motor
109 C_dme = Drive.Motor.Kt;                     %C matrix of drive motor
        electrical
110
111 A_lme = -Load.Motor.R/Load.Motor.L;        %A matrix of load motor
        electrical
112 B_lme = 1/Load.Motor.L;                    %B matrix of load motor
        electrical
113 E_lme = -Load.Motor.Kw/Load.Motor.L;       %E matrix of load motor
114 C_lme = Load.Motor.Kt;                     %C matrix of load motor
        electrical
115
116 A_sys = [0, 1; 0, -System.b/System.J];     %A matrix of System
        Mechanical
117 B_sys = [0; 1/System.J];                   %B matrix of System
        Mechanical
118 C_sys = [1, 0];                           %C matrix of System
        Mechanical
119
120 %% controller feedback gain matrices
121 KK1_DP = [Controller.K11, Controller.K12];
122 KK2_DP = Controller.K2;
123
124 KK1_DT = Drive.PCE.K1;
125 KK2_DT = Drive.PCE.K2;
126
127 KK1_LT = Load.PCE.K1;
128 KK2_LT = Load.PCE.K2;
129 %% data converters
130 Drive.CurrentSensor.R = 0.0015;
131 Drive.CurrentSensor.G = 60;
132 Drive.CurrentSensor.Bits = 12;
```

```
133 Drive.CurrentSensor.VREF = 1.5;
134 Drive.CurrenSensor.MaxVoltage = 3.3;
135 Drive.CurrentSensor.MinVoltage = 0;
136
137 Load.CurrentSensor.R = 0.0015;
138 Load.CurrentSensor.G = 60;
139 Load.CurrentSensor.Bits = 12;
140 Load.CurrentSensor.VREF = 1.5;
141 Load.CurrenSensor.MaxVoltage = 3.3;
142 Load.CurrentSensor.MinVoltage = 0;
143
144 %% time grid
145 T = 1/Controller.Frequency;
146 h = T/100;
147 %t = 0:h:Controller.Tau * 20;
148 t = 0:h:1;
149
150 %% reference input
151 %r = deg2rad(22.5);
152 %r = 0.785398;
153 r = deg2rad(45);
154 %TC = Controller.Tau * 10;
155 TC = inf;
156
157 %% preallocation
158 V_dm = zeros(1,length(t));        %Drive Motor Voltage
159 TC_dm = zeros(1,length(t));       %Drive Torque Command
160 I_dm = zeros(1,length(t));        %Drive Motor Current
161
162 V_lm = zeros(1,length(t));        %Load Motor Voltage
163 TC_lm = zeros(1,length(t));       %Load Torque Command
164 I_lm = zeros(1,length(t));        %Load Motor Current
165 I_ls = zeros(1,length(t));
166 I_ds = zeros(1,length(t));
167 X_sys = zeros(2,length(t));       %System Position and Velocity
168 R = zeros(1,length(t));
169
170 %% initial conditions
171 x_sys = [System.Initial_Position;System.Initial_Speed];
172 xhat_P = [0;0];
173 sigma_DP = 0;
174 xhat_C = 0;
175 sigma_DC = 0;
176 sigma_LC = 0;
177 Vd = 0;
178 Vl = 0;
179 P_sys = x_sys(1);
180 P_dm = P_sys/Drive.Geartrain.GearRatio*Drive.enable;
181 P_lm = -P_sys/Drive.Geartrain.GearRatio*Drive.enable;
182 omega_dm = 0;
183 omega_lm = 0;
184 x_dm = 0;
185 x_lm = 0;
186 store_D = 0;
```

```
187 store_L = 0;
188 store_LL = 0;
189
190 Pddotdot_sys = 0;
191 store_sys = 0;
192 omegadot_ls = 0;
193
194 IN_dm = [P_dm,P_dm,P_dm];
195 OUT_dm = [0,0,0];
196
197 IN_lm = [P_lm,P_lm,P_lm];
198 OUT_lm = [0,0,0];
199 IN_llm = [0,0,0];
200 OUT_llm = [0,0,0];
201
202 Vin_DS = [1.5,1.5,1.5];
203 Vout_DS = [1.5,1.5,1.5];
204
205 Vin_LS = [1.5,1.5,1.5];
206 Vout_LS = [1.5,1.5,1.5];
207
208 testRE = zeros(1,length(t));
209 testOM = zeros(1,length(t));
210
211 %% simulation time loop
212 for n = 0:length(t)-1
213     if mod(n*h,TC) == 0 %% Change command every 0.5 sec
214         %r = deg2rad(45 + (-45-45)*rand(1,1));
215         r = -r;
216     end
217
218     % microcontroller code (discrete-time update)
219     if mod(n*h,T) == 0
220         %r = r_mag*sin(2*pi*TC*t(n+1));
221         % controller output to actuator
222         V_da = (24/1000)*round(Vd/(24/1000));
223         V_la = (24/1000)*round(Vl/(24/1000));
224
225         % controller input from sensor
226         P_DS = (2*pi/4096)*round(P_dm/(2*pi/4096));
227         %P_LS = (2*pi/4096)*round(P_lm/(2*pi/4096));
228         P_LS = P_lm;
229
230         [I_DS, Vin_DS, Vout_DS] = CurrentSensor(Drive.CurrentSensor.R
                ,...
231                                 Drive.CurrentSensor.G,...
232                                 Drive.CurrentSensor.Bits,...
233                                 Drive.CurrentSensor.VREF,...
234                                 Drive.CurrenSensor.MaxVoltage,...
235                                 Drive.CurrentSensor.MinVoltage,...
236                                 Vin_DS,Vout_DS,x_dm);
237         [I_LS, Vin_LS, Vout_LS] = CurrentSensor(Load.CurrentSensor.R,...
238                                 Load.CurrentSensor.G,...
239                                 Load.CurrentSensor.Bits,...
```

```
240                             Load.CurrentSensor.VREF,...
241                             Load.CurrenSensor.MaxVoltage,...
242                             Load.CurrentSensor.MinVoltage,...
243                             Vin_LS,Vout_LS,x_lm);
244
245         % controller velocity estimation
246
247
248         IN_dm(1) = P_dm;
249         [IN_dm, OUT_dm] = RateEstimation(IN_dm,OUT_dm);
250         Pddot_ds = OUT_dm(1);
251
252         IN_lm(1) = P_lm;
253         [IN_lm, OUT_lm] = RateEstimation(IN_lm,OUT_lm);
254         Pddot_ls = OUT_lm(1);
255
256         IN_llm(1) = Pddot_ls;
257         [IN_llm, OUT_lm] = RateEstimation(IN_llm,OUT_llm);
258         Pddotdot_ls = OUT_llm(1);
259
260         %end
261         testRE(:,n+1) = Pddot_ds;
262         testOM(:,n+1) = omega_dm;
263
264         % compute Drive controller output
265         TCd = -KK1_DP*[P_DS; Pddot_ds]-KK2_DP*sigma_DP;
                                          %Drive Position
266         Vd = -KK1_DT*Drive.Motor.Kt*I_DS-KK2_DT*sigma_DC;
                                        %Drive Current
267
268         % Update Max Torque
269         Drive.Max_Torque = Drive.Motor.Kw/Drive.Motor.R*System.
                Max_Voltage - Drive.Motor.Kw*Drive.Motor.Kt/Drive.Motor.R*
                Pddot_ds;
270         Load.Max_Torque = Load.Motor.Kw/Load.Motor.R*System.Max_Voltage
                - Load.Motor.Kw*Load.Motor.Kt/Load.Motor.R*Pddot_ls;
271         % update controller state variables
272         if abs(Vd) > System.Max_Voltage, Vd = System.Max_Voltage*sign(Vd
                );
273         else, sigma_DC = sigma_DC+T*(Drive.Motor.Kt*I_DS-TCd);
                                    %Anti-windup
274         end
275
276         if abs(TCd) > Drive.Max_Torque/Drive.Geartrain.GearRatio, TCd =
                Drive.Max_Torque/Drive.Geartrain.GearRatio*sign(TCd);
277         else, sigma_DP = sigma_DP+T*(P_DS-r/Drive.Geartrain.GearRatio);
                        %Anti-windup
278         end
279
280         % compute Load controller output
281          TCl = Load.enable*...
282                 (Load.Geartrain.GearRatio)*...
283                 (sin(-P_LS*Load.Geartrain.GearRatio)*Load.Model.F*Load.
                        Model.dcp...
```

```
284                 +sin(-P_LS*Load.Geartrain.GearRatio)*Load.Model.a*Load.
                        Model.m*Load.Model.dcg...
285                 +(-Pddotdot_ls*Load.Model.J));
286
287         %if abs(TCl) > Load.Max_Torque/Load.Geartrain.GearRatio, TCl =
                Load.Max_Torque/Load.Geartrain.GearRatio*sign(TCl);
288         %end
289         Vl = (-KK1_LT*Drive.Motor.Kt*I_LS-KK2_LT*sigma_LC);
                                        %Load Current
290
291         % update controller state variables
292         if abs(Vl) > System.Max_Voltage, Vl = System.Max_Voltage*sign(Vl
                );
293         else, sigma_LC = sigma_LC+T*(Drive.Motor.Kt*I_LS-TCl);
                                        %Anti-windup
294         end
295     end
296
297     % store results (only for analysis)
298     X_sys(:,n+1) = x_sys;
299     R(:, n+1) = r;
300
301     V_dm(:,n+1) = V_da;
302     TC_dm(:,n+1) = TCd;
303     I_dm(:,n+1) = x_dm;
304     I_ds(:,n+1) = I_DS;
305
306     V_lm(:,n+1) = V_la;
307     TC_lm(:,n+1) = TCl;
308     I_lm(:,n+1) = x_lm;
309     I_ls(:,n+1) = I_LS;
310     %SYSMT(:, n+1) = System.Max_Torque;
311
312     % plant physics (continuous-time update)
313     %Drive elec
314     x_dm = x_dm+h*(A_dme*x_dm+B_dme*Vd+E_dme*omega_dm); %w_dm =
            sys_omega
315     T_dm = C_dme*x_dm;
316
317     %Load elec
318     x_lm = x_lm+h*(A_lme*x_lm+B_lme*Vl+E_lme*omega_lm); %w_lm = -
            sys_omega
319     T_lm = C_lme*x_lm;
320
321     %Geartrain (Forward)
322     T_sys =  T_dm/Drive.Geartrain.GearRatio - ...
323          T_lm/Load.Geartrain.GearRatio*Load.enable;
324
325     %mech
326     x_sys = x_sys+h*(A_sys*x_sys+B_sys*T_sys);
327     P_sys = C_sys*x_sys;
328
329     %Geartrain (Reverse)
330     omega_dm = x_sys(2)/Drive.Geartrain.GearRatio;
```

```
331         omega_lm = -x_sys(2)/Load.Geartrain.GearRatio*Load.enable;
332
333         P_dm = P_sys/Drive.Geartrain.GearRatio;
334         P_lm = -P_sys/Load.Geartrain.GearRatio*Load.enable;
335
336         Pddotdot_sys = (x_sys(2)-store_sys)/T;
337         store_sys = x_sys(2);
338         omegadot_ls = -Pddotdot_sys/Load.Geartrain.GearRatio*Load.enable;
339     end
340
341     %% Post Processing
342     ResponseInfo = lsiminfo(X_sys(1,:),t,r);
343
344     %% closed-loop response plots
345     % close all;
346     % figure()
347     % subplot(6,1,1)
348     % hold on;
349     % plot(t,rad2deg(X_sys(1,:)));
350     % plot(t,rad2deg(R),'--');
351     % plot(t,ones(1,length(t))*rad2deg(System.Max_Position),'k-')
352     % plot(t,ones(1,length(t))*-rad2deg(System.Max_Position),'k-')
353     % xlabel('t [s]'), ylabel('\theta(t) [deg]'), title('System Position')
354     % %if(System.Max_Position ~= inf), ylim([-rad2deg(System.Max_Position)
            rad2deg(System.Max_Position)]), xlim([0 t(end)]);
355     % %else, ylim([-max([max(abs(rad2deg(X_sys(1,:)))),max(abs(rad2deg(R)))
            ]) max([max(abs(rad2deg(X_sys(1,:)))),max(abs(rad2deg(R)))])), xlim
            ([0 t(end)]); end
356     % legend('Position', 'Command');
357     %
358     % subplot(6,1,2)
359     % hold on;
360     % plot(t,X_sys(2,:));
361     % xlabel('t [s]'), ylabel('\omega(t) [rad/s]'), title('System Speed')
362     % %xlim([0 t(end)]);
363     %
364     % subplot(6,1,3)
365     % hold on;
366     % plot(t,V_dm);
367     % %plot(t,ones(1,length(t))*System.Max_Voltage,'k-')
368     % %plot(t,ones(1,length(t))*-System.Max_Voltage,'k-')
369     % xlabel('t [s]'), ylabel('v(t) [V]'), title('Drive Motor Voltage')
370     % %ylim([-System.Max_Voltage System.Max_Voltage]), xlim([0 t(end)]);
371     %
372     % subplot(6,1,4)
373     % hold on;
374     % plot(t,I_dm(1,:))
375     % plot(t,TC_dm/Drive.Motor.Kt,'--')
376     % %plot(t,SYSMT/Drive.Motor.Kt,'k-')
377     % %plot(t,-SYSMT/Drive.Motor.Kt,'k-')
378     % xlabel('t [s]'), ylabel('i(t) [A]'), title('Drive Motor Current')
379     % legend('Current', 'Command');
380     %
381     % subplot(6,1,5)
```

```
382 % hold on;
383 % plot(t,V_lm)
384 % %plot(t,ones(1,length(t))*System.Max_Voltage,'k-')
385 % %plot(t,ones(1,length(t))*-System.Max_Voltage,'k-')
386 % xlabel('t [s]'), ylabel('v(t) [V]'), title('Load Motor Voltage')
387 %
388 % subplot(6,1,6)
389 % hold on;
390 % plot(t,I_lm(1,:));
391 % plot(t,TC_lm/Drive.Motor.Kt,'--');
392 % xlabel('t [s]'), ylabel('i(t) [A]'), title('Load Motor Current')
393 % legend('Current', 'Command');
394 %
395 %% C Code Plots
396 % figure()
397 % %subplot(2,1,1)
398 % hold on;
399 % title('System Position');
400 % xlabel('Time (ms)');
401 % ylabel('Position (deg)');
402 % xlim([CDATA_t(1) CDATA_t(end)].*1000);
403 % plot(CDATA_t.*1000,rad2deg(CDATA_SYS_P));
404 % plot(t.*1000,rad2deg(X_sys(1,:)));
405 % plot(t.*1000,rad2deg(R),'--');
406 % legend('Experimental Data','Simulation','Position Command');
407 %
408 % figure()
409 % %subplot(2,1,1)
410 % hold on;
411 % title('Drive Motor Current');
412 % xlabel('Time (ms)');
413 % ylabel('Current (A)');
414 % xlim([CDATA_t(1) CDATA_t(end)].*1000);
415 % plot(CDATA_t.*1000,CDATA_I_dm);
416 % plot(t.*1000,I_dm);
417 % legend('Experimental Data','Simulation');
418 %
419 % figure()
420 % %subplot(2,1,1)
421 % hold on;
422 % title('Load Motor Current');
423 % xlabel('Time (ms)');
424 % ylabel('Current (A)');
425 % xlim([CDATA_t(1) CDATA_t(end)].*1000);
426 % plot(CDATA_t.*1000,CDATA_I_lm);
427 % plot(t.*1000,I_lm);
428 % legend('Experimental Data','Simulation');
429
430 %C_Code_Plots_STEP();
```

B.2 RateEstimation.m

```
function [IN, OUT] = RateEstimation(IN, OUT)

    T = 1/30000;
    A = 1/(T*1.17376);

    OUT(3) = OUT(2);
    OUT(2) = OUT(1);
    OUT(1) = IN(1)*A ...
             +IN(2)*0 ...
             +IN(3)*-A ...
             -OUT(2)*0.611 ...
             -OUT(3)*0.0932;

    IN(3) = IN(2);
    IN(2) = IN(1);
end
```

B.3 CurrentSensor.m

```
function [ out, INARRAY, OUTARRAY ] = CurrentSensor( R,G,bits,VREF,Max,
    Min,INARRAY,OUTARRAY, in )
      V = VREF+((in*0.0015)*60);
      INARRAY(1) = V;
      OUTARRAY(1) = V;

      VFLTR = OUTARRAY(1);

      V_ADC = ((Max-Min)/2^bits)*round(VFLTR/((Max-Min)/2^bits));
      if (V_ADC > Max), V_ADC = Max; end
      if (V_ADC < Min), V_ADC = Min; end

      out = (V_ADC-VREF)/(G*R);

end
```

APPENDIX C

MICROCONTROLLER CODE

C.1 DMTB.h (main.h)

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB.h
/**********************************/

#ifndef DMTB_H_
#define DMTB_H_

#include <stdbool.h>
#include <stdint.h>
#include "F2806x_Device.h"

#include "Math.h"

#include "DMTB_GlobalVariableDefs.h"

#include "DMTB_ADC.h"
#include "DMTB_GPIO.h"
#include "DMTB_LED.h"
#include "DMTB_Motor.h"
#include "DMTB_PWM.h"
#include "DMTB_Filtering.h"
#include "DMTB_Encoder.h"

interrupt void cpu_timer0_isr(void);
interrupt void adc1_isr(void);

#endif /* DMTB_H_ */
```

C.2 DMTB.c (main.c)

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB.c
/**********************************/

#include "DMTB.h"

#define PI (float)3.1415
#define TVE 600
#define A ((float)TVE/(float)1.17376)

#define VMAX_CONTROLLER ((float)0.95*(float)12)
#define VMIN_CONTROLLER -VMAX_CONTROLLER

//#define VMAX_LOAD ((float)0.95*(float)6)
#define VMAX_LOAD 6
#define VMIN_LOAD -VMAX_LOAD

#define TSTOP             (float)0.25
#define TREF              (float)0.5
//#define REFCMD            -0.785398 //45 deg
#define REFCMD            -0.349066
#define SamplePeriod      TSTOP/(float)SIZE

Uint32 div = 0;
float tempsin = 0;

void main(void)
{
    EALLOW;
    SysCtrlRegs.WDCR = 0x68;  // Disable Watchdog

    SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL = 0;
    if(SysCtrlRegs.PLLSTS.bit.MCLKSTS == 0)
    {
        if((SysCtrlRegs.PLLSTS.bit.DIVSEL == 2)||(SysCtrlRegs.PLLSTS.bit
            .DIVSEL == 3))
        {
            SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
        }

        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
        SysCtrlRegs.PLLCR.bit.DIV = 9;
        while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)
        {
            //Do nothing
```

```
50          }
51          SysCtrlRegs.PLLSTS.bit.MCLKOFF  = 0;
52          SysCtrlRegs.PLLSTS.bit.DIVSEL = 3;   //(Set SysClk to 90MHz)
53      }
54
55      DINT;
56      IER = 0x0000;
57      IFR = 0x0000;
58
59      //Timer 0 Setup
60      CpuTimer0Regs.TPR.bit.TDDR = 0; //Configure Timer0 Interrupt
61      CpuTimer0Regs.TPRH.bit.TDDRH = 0;
62      CpuTimer0Regs.PRD.all = 3000; //Divide SysCLK by 3000 (30kHz)
63      CpuTimer0Regs.TCR.bit.TSS = 1;
64      CpuTimer0Regs.TCR.bit.TRB = 1;
65      CpuTimer0Regs.TCR.bit.TIE = 1;
66
67      Drive_Position_Tau   = 0.025;
68      Drive_Current_Tau    = 0.000175;
69      Load_Current_Tau     = 0.00175;
70
71      EDIS;
72      System_J = 3.4400e-07;
73
74      Drive_Position_K11   = (DriveMotor_Gearratio*System_J*(float)3*((
            float)1/Drive_Position_Tau)*((float)1/Drive_Position_Tau));
75      Drive_Position_K12   = (DriveMotor_Gearratio*System_J*(float)3*((
            float)1/Drive_Position_Tau));
76      Drive_Position_K2    = (DriveMotor_Gearratio*System_J*((float)1/
            Drive_Position_Tau)*((float)1/Drive_Position_Tau)*((float)1/
            Drive_Position_Tau));
77
78      Drive_Current_K1     = ((float)2*DriveMotor_L*((float)1/
            Drive_Current_Tau) - DriveMotor_R)/DriveMotor_Kt;
79      Drive_Current_K2     = (((float)1/Drive_Current_Tau)*((float)1/
            Drive_Current_Tau)*(DriveMotor_L/DriveMotor_Kt));
80
81      Load_Current_K1      = ((float)2*LoadMotor_L*((float)1/
            Load_Current_Tau) - LoadMotor_R)/LoadMotor_Kt;
82      Load_Current_K2      = (((float)1/Load_Current_Tau)*((float)1/
            Load_Current_Tau)*(LoadMotor_L/LoadMotor_Kt));
83      EALLOW;
84
85      GPIO_INIT();
86      MOTOR_DISABLE();
87
88      EDIS;
89      ReadSSIData();
90      Drive_Position = (2*PI*(((float)Drive_Data_Hex)/4069))-PI;
91      sys_p = Drive_Position*DriveMotor_Gearratio;
92      Load_Position = -sys_p/LoadMotor_Gearratio;
93
94      FLTR_InitFilter(&VE_DRIVE_OMEGA, A, 0, -A, 1, 0.611, 0.0932,
            Drive_Position);
```

```
95      FLTR_InitFilter(&VE_LOAD_OMEGA, A, 0, -A, 1, 0.611, 0.0932,
            Load_Position);
96      FLTR_InitFilter(&VE_LOAD_OMEGADOT, A, 0, -A, 1, 0.611, 0.0932, 0);
97      EALLOW;
98
99      ADC_INIT1();
100
101     PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
102
103     PieVectTable.TINT0 = &cpu_timer0_isr;
104     PieVectTable.ADCINT1 = &adc1_isr;
105
106     PieCtrlRegs.PIEIER1.bit.INTx1 = 1;   //ADC1
107     PieCtrlRegs.PIEIER1.bit.INTx7 = 1;   //Enable CPU timer 1
108
109     IER |= M_INT1;
110
111     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
112
113     CpuTimer0Regs.TCR.bit.TSS = 0;
114     SysCtrlRegs.WDCR = 0x28;  // Re-Enable Watchdog Timer After Setup
115     EINT; // Enable Global interrupt INTM
116
117     ADC_INIT2();
118     PWM_INIT();
119
120     EDIS;
121
122     LED_ALL_OFF();
123
124     while (1)
125     {
126         // Do Nothing
127     }
128 }
129
130 interrupt void cpu_timer0_isr(void)
131 {
132     EALLOW;
133     SysCtrlRegs.WDKEY = 0x55;
134     SysCtrlRegs.WDKEY = 0xAA;
135     EDIS;
136
137     if (MTREN == 0)
138     {
139         EPwm1Regs.CMPA.half.CMPA = HALFPWM;
140         EPwm2Regs.CMPA.half.CMPA = HALFPWM;
141
142         EPwm3Regs.CMPA.half.CMPA = HALFPWM;
143         EPwm4Regs.CMPA.half.CMPA = HALFPWM;
144
145         ADC0_Offset = AdcResult.ADCRESULT0;
146         ADC1_Offset = AdcResult.ADCRESULT1;
147         ADC2_Offset = AdcResult.ADCRESULT2;
```

```
148         ADC3_Offset = AdcResult.ADCRESULT3;
149
150         ReadSSIData();
151         Drive_Position_Current = (2*PI*(((float)Drive_Data_Hex)/4069))-
                PI;
152         if ((Drive_Position_Prev > PI/(float)2)&&(Drive_Position_Current
                < -PI/(float)2)) {rev++;}
153         if((Drive_Position_Prev < -(PI/(float)2))&&(
                Drive_Position_Current > (PI/(float)2))) {rev--;}
154         Drive_Position = rev*PI*(float)2 + Drive_Position_Current;
155         sys_p = Drive_Position*DriveMotor_Gearratio;
156         Load_Position = -sys_p/LoadMotor_Gearratio;
157         Drive_Position_Prev = Drive_Position_Current;
158
159         if (div >= 30000/TVE)
160         {
161             Drive_Omega = FLTR_2OrderFilter(&VE_DRIVE_OMEGA,
                    Drive_Position);
162             Load_Omega = FLTR_2OrderFilter(&VE_LOAD_OMEGA, Load_Position
                    );
163             Load_OmegaDot = FLTR_2OrderFilter(&VE_LOAD_OMEGADOT,
                    Load_Omega);
164             div = 0;
165         }
166         else
167         {
168             div++;
169         }
170
171         MOTOR_DISABLE();
172     }
173     else
174     {
175         MOTOR_ENABLE();
176         Load_Voltage = 0;
177         // Write out Previous Voltage Command
178         EPwm1Regs.CMPA.half.CMPA = FULLPWM*(0.5+(Drive_Voltage/24));
179         EPwm2Regs.CMPA.half.CMPA = FULLPWM*(0.5+(Drive_Voltage/24));
180         EPwm3Regs.CMPA.half.CMPA = FULLPWM*(0.5+(Load_Voltage/24));
181         EPwm4Regs.CMPA.half.CMPA = FULLPWM*(0.5+(Load_Voltage/24));
182
183         // Update Position Command
184         if ((float)k/(float)30000 <= TREF){r = -REFCMD;}
185         else if ((float)k/(float)30000 <= (float)2*TREF){r = REFCMD;}
186         else {k = 0;}
187
188         ADC_Voltage1 = (((float)AdcResult.ADCRESULT0 - (float)
                ADC0_Offset)-((float)AdcResult.ADCRESULT1 - (float)
                ADC1_Offset))/(float)2*((float)3.3/(float)4096); //average 2
                 Ints and Calc Voltage
189         ADC_Voltage2 = (((float)AdcResult.ADCRESULT2 - (float)
                ADC2_Offset)-((float)AdcResult.ADCRESULT3 - (float)
                ADC3_Offset))/(float)2*((float)3.3/(float)4096); //average 2
                 Ints and Calc Voltage
```

```
190
191         // Sample Current
192         Drive_Current = ((ADC_Voltage1/(float)0.0015/(float)60)); //
                Current Calculation
193         Load_Current = ((ADC_Voltage2/(float)0.0015/(float)60)); //
                Current Calculation
194
195         // Sample Encoder
196         ReadSSIData();
197         Drive_Position_Current = (2*PI*(((float)Drive_Data_Hex)/4069))-
                PI;
198         if ((Drive_Position_Prev > PI/(float)2)&&(Drive_Position_Current
                < -PI/(float)2)) {rev++;}
199         if((Drive_Position_Prev < -(PI/(float)2))&&(
                Drive_Position_Current > (PI/(float)2))) {rev--;}
200         Drive_Position = rev*PI*(float)2 + Drive_Position_Current;
201         sys_p = Drive_Position*DriveMotor_Gearratio;
202         Load_Position = -sys_p/LoadMotor_Gearratio;
203         Drive_Position_Prev = Drive_Position_Current;
204
205         // Velocity Estimation
206         if (div >= 30000/TVE)
207         {
208             Drive_Omega = FLTR_2OrderFilter(&VE_DRIVE_OMEGA,
                    Drive_Position);
209             Load_Omega = FLTR_2OrderFilter(&VE_LOAD_OMEGA, Load_Position
                    );
210             Load_OmegaDot = FLTR_2OrderFilter(&VE_LOAD_OMEGADOT,
                    Load_Omega);
211             div = 0;
212         }
213         else
214         {
215             div++;
216         }
217
218         // Position Control Loop
219         Drive_TorqueCommand = (-Drive_Position_K11*Drive_Position) + (-
                Drive_Position_K12*Drive_Omega) + (-Drive_Position_K2*
                Drive_Position_Sigma);
220         Drive_Voltage = (-Drive_Current_K1*DriveMotor_Kt*Drive_Current)
                +(-Drive_Current_K2*Drive_Current_Sigma);
221
222
223         if (Drive_Voltage > VMAX_CONTROLLER) {Drive_Voltage =
                VMAX_CONTROLLER;}
224         else if (Drive_Voltage < VMIN_CONTROLLER) {Drive_Voltage =
                VMIN_CONTROLLER;}
225         else {Drive_Current_Sigma = Drive_Current_Sigma + ((float)1/(
                float)30000)*(DriveMotor_Kt*Drive_Current-
                Drive_TorqueCommand);}
226
227         // Current Control Loop
```

```
228         TorqueCommandMax = (DriveMotor_Kw/DriveMotor_R*VMAX_CONTROLLER -
                DriveMotor_Kw*DriveMotor_Kt/DriveMotor_R*Drive_Omega)/
            DriveMotor_Gearratio;
229
230         if (Drive_TorqueCommand > TorqueCommandMax) {Drive_TorqueCommand
                = TorqueCommandMax;}
231         else if (Drive_TorqueCommand < -TorqueCommandMax) {
            Drive_TorqueCommand = -TorqueCommandMax;}
232         else {Drive_Position_Sigma = Drive_Position_Sigma + ((float)1/(
            float)30000)*(Drive_Position-r/DriveMotor_Gearratio);}
233
234         // Load Model
235         tempsin = sin(sys_p);
236         Load_TorqueCP = sin(sys_p)*LoadModel_F*LoadModel_CP; //Change
237         Load_TorqueCG = sin(sys_p)*LoadModel_a*LoadModel_m*LoadModel_CG;
                //Change
238         Load_TorqueInertia = -Load_OmegaDot*LoadModel_J;
239
240         Load_TorqueCommand = -(Load_TorqueCG + Load_TorqueCP +
            Load_TorqueInertia)*DriveMotor_Gearratio;
241         Load_Voltage = (-Load_Current_K1*LoadMotor_Kt*Load_Current)+(-
            Load_Current_K2*Load_Current_Sigma); //Change
242
243         if (Load_Voltage > VMAX_LOAD) {Load_Voltage = VMAX_LOAD;} //
            Change
244         else if (Load_Voltage < VMIN_LOAD) {Load_Voltage = VMIN_LOAD;}
            //Change
245         else {Load_Current_Sigma = Load_Current_Sigma + ((float)1/(float
            )30000)*(LoadMotor_Kt*Load_Current-Load_TorqueCommand);}
246
247         if (i < SIZE)
248         {
249             if (((float)i*(float)TSTOP/(float)200) <= ((float)j/(float)
                    30000))
250             {
251                 t[i] = SamplePeriod*i;
252                 V_dm[i]  = Drive_Voltage;
253                 V_lm[i]  = Load_Voltage;
254                 TC_dm[i] = Drive_TorqueCommand;
255                 TC_lm[i] = Load_TorqueCommand;
256                 SYS_P[i] = sys_p;
257                 SYS_O[i] = Drive_Omega*DriveMotor_Gearratio;
258                 I_dm[i]  = Drive_Current;
259                 I_lm[i]  = Load_Current;
260                 R[i]     = r;
261                 i++;
262             }
263             j++;
264         }
265         k++;
266     }
267     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
268 }
269
```

```
270 interrupt void adc1_isr(void)
271 {
272     GpioDataRegs.GPBSET.bit.GPIO33 = 1;
273
274     GpioDataRegs.GPBCLEAR.bit.GPIO33 = 1;
275
276     AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
277     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;     // Acknowledge interrupt
            to PIE
278 }
```

C.3 DMTB_ADC.h

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_ADC.h
/**********************************/

#ifndef DMTB_ADC_H_
#define DMTB_ADC_H_

void ADC_INIT1(void);
void ADC_INIT2(void);

#endif /* DMTB_ADC_H_ */
```

C.4 DMTB_ADC.c

```
/***********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_ADC.c
/***********************************/

#include "DMTB.h"

void ADC_INIT1(void)
{
    Uint32 i = 0;
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;

    asm(" NOP");
    asm(" NOP");
    asm(" NOP");

    AdcRegs.ADCCTL1.bit.ADCBGPWD = 1;
    AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;
    AdcRegs.ADCCTL1.bit.ADCPWDN = 1;
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;

    for(i = 0; i < 90000; i++)
    {
        asm(" NOP");
    } // delay 1ms @ 90 MHz)

    AdcRegs.ADCCTL2.bit.CLKDIV2EN = 1;
    AdcRegs.ADCCTL2.bit.CLKDIV4EN = 0;

    for(i = 0; i < 90000; i++)
    {
        asm(" NOP");
    } // delay 1ms @ 90 MHz)
}

void ADC_INIT2(void)
{
    AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1;  // Enable non-overlap mode
    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    //AdcRegs.ADCCTL1.bit.VREFLOCONV = 0;
    //AdcRegs.ADCCTL1.bit.TEMPCONV = 0;

    AdcRegs.INTSEL1N2.bit.INT1E     = 1;    // Enabled ADCINT1
    AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;
```

```
50      AdcRegs.INTSEL1N2.bit.INT1SEL    = 3;     // setup EOC3 to trigger
            ADCINT1 to fire
51
52      //AdcRegs.SOCPRICTL.bit.SOCPRIORITY = 4;     // SOC0 - SOC3 are high
            priority
53
54      AdcRegs.ADCSOC0CTL.bit.CHSEL     = 0x04;     // set SOC0 channel
            select to ADCINA4
55      AdcRegs.ADCSOC1CTL.bit.CHSEL     = 0x00;     // set SOC1 channel
            select to ADCINA0
56      AdcRegs.ADCSOC2CTL.bit.CHSEL     = 0x09;     // set SOC2 channel
            select to ADCINB1
57      AdcRegs.ADCSOC3CTL.bit.CHSEL     = 0x0D;     // set SOC3 channel
            select to ADCINB5
58
59      AdcRegs.ADCSOC0CTL.bit.ACQPS     = 6;     // set SOC0 S/H Window to 7
            ADC Clock Cycles, (6 ACQPS plus 1)
60      AdcRegs.ADCSOC1CTL.bit.ACQPS     = 6;     // set SOC1 S/H Window to 7
            ADC Clock Cycles, (6 ACQPS plus 1)
61      AdcRegs.ADCSOC2CTL.bit.ACQPS     = 6;     // set SOC2 S/H Window to 7
            ADC Clock Cycles, (6 ACQPS plus 1)
62      AdcRegs.ADCSOC3CTL.bit.ACQPS     = 6;     // set SOC3 S/H Window to 7
            ADC Clock Cycles, (6 ACQPS plus 1)
63
64
65      AdcRegs.ADCSOC0CTL.bit.TRIGSEL   = 0x05;     // set SOC0 start trigger
            on EPWM1A, ADCSOCA
66      AdcRegs.ADCSOC1CTL.bit.TRIGSEL   = 0x05;     // set SOC1 start trigger
            on EPWM1A, ADCSOCA
67      AdcRegs.ADCSOC2CTL.bit.TRIGSEL   = 0x05;     // set SOC2 start trigger
            on EPWM1A, ADCSOCA
68      AdcRegs.ADCSOC3CTL.bit.TRIGSEL   = 0x05;     // set SOC3 start trigger
            on EPWM1A, ADCSOCA
69
70      /*
71      AdcRegs.ADCSOC0CTL.bit.TRIGSEL   = 0x1;     // set SOC0 start trigger
            on Timer 0
72      AdcRegs.ADCSOC1CTL.bit.TRIGSEL   = 0x1;
73      AdcRegs.ADCSOC2CTL.bit.TRIGSEL   = 0x1;
74      AdcRegs.ADCSOC3CTL.bit.TRIGSEL   = 0x1;
75      */
76  }
```

C.5 DMTB_Encoder.h

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_Encoder.h
/**********************************/

#ifndef DMTB_ENCODER_H_
#define DMTB_ENCODER_H_

void ReadSSIData(void);

#endif /* DMTB_ENCODER_H_ */
```

C.6 DMTB_Encoder.c

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_Encoder.c
/**********************************/

#include "DMTB.h"

void ReadSSIData(void)
{
    //Uint16 z = 0;

    Data1_B = 0;
    Data1_G = 0;

    //bool test[13] = {1,1,0,1,1,1,1,1,0,0,0,1,0};

    // falling edge on clock port
    GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
    //GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; //Load
    /*
    for(z = 0; z < 10; z++)
    {
        asm(" NOP");
    }
    */
    for (bitCount = 0; bitCount < 13; bitCount++)
    {
        // rising edge on clock port, data changes
        GpioDataRegs.GPASET.bit.GPIO10 = 1;
        //GpioDataRegs.GPASET.bit.GPIO11 = 1; //Load

        // left-shift the current result
        Data1_B = (Data1_B << 1);
        Data1_G = (Data1_G << 1);
        /*
        for(z = 0; z < 10; z++)
        {
            asm(" NOP");
        }
        */
        // falling edge on clock port
        GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
        //GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; //Load

        Data1_G |= GpioDataRegs.GPADAT.bit.GPIO8;
        //Data1_G |= GpioDataRegs.GPADAT.bit.GPIO9; //Load
```

```
51          //Data1_G |= test[bitCount];
52
53          // read the port data
54          if(Data1_G & 0x01)
55          {
56              Data1_B |= !((Data1_B & 0x02)&&(0x02));
57          }
58          else
59          {
60              Data1_B |= ((Data1_B & 0x02)&&(0x02));
61          }
62      }
63
64      // rising edge on clock port, data changes
65      GpioDataRegs.GPASET.bit.GPIO10 = 1;
66      //GpioDataRegs.GPASET.bit.GPIO11 = 1; //Load
67
68      // Throw away last bit
69      Drive_Data_Hex = Data1_B >> 1;
70  }
```

C.7 DMTB_Filtering.h

```
1  /*********************************/
2  /* Jacob Sobering
3  /* Master's Thesis Research
4  /* Georgia Institute of Technology
5  /* 2017-2018
6  /*
7  /* DMTB_Filtering.h
8  /*********************************/
9
10 #ifndef LCMD_FILTERING_H_
11 #define LCMD_FILTERING_H_
12
13 void FLTR_InitFilter(FILTER_ORDER2 *filter, float A0, float A1, float A2
       , float B0, float B1, float B2, float X0);
14 float FLTR_2OrderFilter(FILTER_ORDER2*, float in);
15
16 #endif /* LCMD_FILTERING_H_ */
```

C.8 DMTB_Filtering.c

```c
/*********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_Filtering.c
/*********************************/

#include "DMTB.h"

void FLTR_InitFilter(FILTER_ORDER2 *filter, float A0, float A1, float A2
    , float B0, float B1, float B2, float X0)
{
    filter->y0 = 0;
    filter->y1 = 0;
    filter->y2 = 0;
    filter->x0 = X0;
    filter->x1 = X0;
    filter->x2 = X0;
    filter->a0 = A0;
    filter->a1 = A1;
    filter->a2 = A2;
    filter->b0 = B0;
    filter->b1 = B1;
    filter->b2 = B2;
}

float FLTR_2OrderFilter(FILTER_ORDER2 *filter, float in)
{
    filter->x2 = filter->x1;          //Store previous Inputs
    filter->x1 = filter->x0;
    filter->x0 = in;                  //Pass in new input

    filter->y2 = filter->y1;          //Store previous Outputs
    filter->y1 = filter->y0;
    filter->y0 = ((filter->x0*filter->a0)+(filter->x1*filter->a1)+(
        filter->x2*filter->a2)-(filter->y1*filter->b1)-(filter->y2*
        filter->b2))/filter->b0;   //Filter calcs

    return filter->y0;
}
```

C.9 DMTB_GlobalVariableDefs.h

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_GlobalVariableDefs.h
/**********************************/

#ifndef DMTB_GLOBALVARIABLEDEFS_H_
#define DMTB_GLOBALVARIABLEDEFS_H_

#include "F2806x_Device.h"

#define SIZE 200

#define FULLPWM 500
#define HALFPWM 250

extern Uint8 run;
extern Uint8 MTREN;

extern Uint32 i;
extern Uint32 j;
extern Uint32 k;

extern Uint8 bitCount;
extern Uint16 Data1_G;
extern Uint16 Data1_B;

extern float sys_p;
extern float rev;

typedef struct
{
    float y0;    //output
    float y1;    //previous output
    float y2;    //previous previous output
    float x0;    //input
    float x1;    //previous input
    float x2;    //previous previous input
    float a0;
    float a1;
    float a2;
    float b0;
    float b1;
    float b2;

}FILTER_ORDER2;

```

```
51 extern FILTER_ORDER2 VE_DRIVE_OMEGA;
52 extern FILTER_ORDER2 VE_LOAD_OMEGA;
53 extern FILTER_ORDER2 VE_LOAD_OMEGADOT;
54
55 /* ADC Global Variables */
56 extern Uint16 ADC_Conv1;
57 extern Uint16 ADC_Conv2;
58 extern Uint16 ADC_Conv3;
59 extern Uint16 ADC_Conv4;
60
61 extern float ADC_Voltage1;
62 extern float ADC_Voltage2;
63
64 /* Drive Global Variables */
65 extern float r;
66 extern float System_J;
67
68 extern float Drive_Voltage;
69 extern float Drive_TorqueCommand;
70 extern float TorqueCommandMax;
71
72 extern Uint32 ADC0_Offset;
73 extern Uint32 ADC1_Offset;
74 extern Uint32 ADC2_Offset;
75 extern Uint32 ADC3_Offset;
76
77 extern float Drive_Current;
78 extern Uint32 Drive_Data_Hex;
79 extern float DP_OFFSET;
80 extern float Drive_Position;
81 extern float Drive_Position_Current;
82 extern float Drive_Position_Prev;
83 extern float Drive_Position_Deg;
84 extern float Drive_Omega;
85
86 extern float DriveMotor_Kt;
87 extern float DriveMotor_Kw;
88 extern float DriveMotor_R;
89 extern float DriveMotor_L;
90 extern float DriveMotor_Gearratio;
91
92 extern float Drive_Position_Sigma;
93 extern float Drive_Current_Sigma;
94
95 extern float Drive_Position_Tau;
96 extern float Drive_Position_K11;
97 extern float Drive_Position_K12;
98 extern float Drive_Position_K2;
99
100 extern float Drive_Current_Tau;
101 extern float Drive_Current_K1;
102 extern float Drive_Current_K2;
103
104 /* Load Global Variables */
```

```
105 extern float Load_Voltage;
106 extern float Load_TorqueCommand;
107
108 extern float Load_TorqueInertia;
109 extern float Load_TorqueCP;
110 extern float Load_TorqueCG;
111
112 extern float LoadModel_a;
113 extern float LoadModel_m;
114 extern float LoadModel_CG;
115 extern float LoadModel_F;
116 extern float LoadModel_CP;
117 extern float LoadModel_J;
118
119 extern float Load_Current;
120 extern Uint32 Load_Data_Hex;
121 extern float LP_OFFSET;
122 extern float Load_Position;
123 extern float Load_Omega;
124 extern float Load_OmegaDot;
125
126 extern float LoadMotor_Kt;
127 extern float LoadMotor_Kw;
128 extern float LoadMotor_R;
129 extern float LoadMotor_L;
130 extern float LoadMotor_Gearratio;
131
132 extern float Load_Current_Sigma;
133
134 extern float Load_Current_Tau;
135 extern float Load_Current_K1;
136 extern float Load_Current_K2;
137
138 extern float t[SIZE];
139 extern float V_dm[SIZE];
140 extern float V_lm[SIZE];
141 extern float TC_dm[SIZE];
142 extern float TC_lm[SIZE];
143 extern float SYS_P[SIZE];
144 extern float SYS_O[SIZE];
145 extern float I_dm[SIZE];
146 extern float I_lm[SIZE];
147 extern float R[SIZE];
148
149 extern Uint8 LoopCount;
150 #endif /* DMTB_GLOBALVARIABLEDEFS_H_ */
```

C.10 DMTB_GlobalVariableDefs.c

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_GlobalVariableDefs.c
/**********************************/

#include <stdbool.h>
#include <stdint.h>
#include "DMTB_GlobalVariableDefs.h"

Uint8 run = 0;
Uint8 MTREN = 0;

Uint32 i = 0;
Uint32 j = 0;
Uint32 k = 0;

Uint8 bitCount = 0;
Uint16 Data1_G = 0;
Uint16 Data1_B = 0;

float sys_p = 0;
float rev = 0;

FILTER_ORDER2 VE_DRIVE_OMEGA;
FILTER_ORDER2 VE_LOAD_OMEGA;
FILTER_ORDER2 VE_LOAD_OMEGADOT;

/* ADC Global Variables */
Uint16 ADC_Conv1 = 0;
Uint16 ADC_Conv2 = 0;
Uint16 ADC_Conv3 = 0;
Uint16 ADC_Conv4 = 0;

float ADC_Voltage1 = 0;
float ADC_Voltage2 = 0;

/* Drive Global Variables */
float r = 0;
float System_J = 0.000000344;

float Drive_Voltage = 0;
float Drive_TorqueCommand = 0;
float TorqueCommandMax = 0;

Uint32 ADC0_Offset = 0;
Uint32 ADC1_Offset = 0;
```

```
51 Uint32 ADC2_Offset = 0;
52 Uint32 ADC3_Offset = 0;
53
54 float Drive_Current = 0;
55 Uint32 Drive_Data_Hex = 0;
56 float DP_OFFSET = 0;
57 float Drive_Position = 0;
58 float Drive_Position_Current = 0;
59 float Drive_Position_Prev = 0;
60 float Drive_Position_Deg = 0;
61 float Drive_Omega = 0;
62
63 float DriveMotor_Kt = 0.00853;
64 float DriveMotor_Kw = 0.00853;
65 float DriveMotor_R = 5.15;
66 float DriveMotor_L = 0.000131;
67 float DriveMotor_Gearratio = (1/(float)6.6);
68
69 float Drive_Position_Sigma = 0;
70 float Drive_Current_Sigma = 0;
71
72 float Drive_Position_Tau    = 0;
73 float Drive_Position_K11    = 0;
74 float Drive_Position_K12    = 0;
75 float Drive_Position_K2     = 0;
76
77 float Drive_Current_Tau     = 0;
78 float Drive_Current_K1      = 0;
79 float Drive_Current_K2      = 0;
80
81 /* Load Global Variables */
82 float Load_Voltage = 0;
83 float Load_TorqueCommand = 0;
84
85 float Load_TorqueInertia = 0;
86 float Load_TorqueCP = 0;
87 float Load_TorqueCG = 0;
88
89 float LoadModel_a = 15;
90 float LoadModel_m = 0.1;
91 float LoadModel_CG = 0.005;
92 float LoadModel_F = 0;
93 float LoadModel_CP = 0;
94 float LoadModel_J = 0.0000025;
95
96 float Load_Current = 0;
97 Uint32 Load_Data_Hex = 0;
98 float LP_OFFSET = 0;
99 float Load_Position = 0;
100 float Load_Omega = 0;
101 float Load_OmegaDot = 0;
102 /*
103 float LoadMotor_Kt = 0.00859;
104 float LoadMotor_Kw = 0.00859;
```

```
105 float LoadMotor_R = 8.694;
106 float LoadMotor_L = 0.000096;
107 float LoadMotor_Gearratio = (1/(float)3.9);
108 */
109 float LoadMotor_Kt = 0.00853;
110 float LoadMotor_Kw = 0.00853;
111 float LoadMotor_R = 5.15;
112 float LoadMotor_L = 0.000131;
113 float LoadMotor_Gearratio = (1/(float)6.6);
114
115 float Load_Current_Sigma = 0;
116
117 float Load_Current_Tau = 0;
118 float Load_Current_K1 = 0;
119 float Load_Current_K2 = 0;
120
121 float t[SIZE] = {0};
122 float V_dm[SIZE]  = {0};
123 float V_lm[SIZE]  = {0};
124 float TC_dm[SIZE] = {0};
125 float TC_lm[SIZE] = {0};
126 float SYS_P[SIZE] = {0};
127 float SYS_O[SIZE] = {0};
128 float I_dm[SIZE]  = {0};
129 float I_lm[SIZE]  = {0};
130 float R[SIZE]     = {0};
131
132 Uint8 LoopCount = 0;
```

C.11 DMTB_GPIO.h

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_GPIO.h
/**********************************/

#ifndef DMTB_GPIO_H_
#define DMTB_GPIO_H_

void GPIO_INIT(void);
void GPIO_ADC(void);
void GPIO_LED(void);
void GPIO_PWM(void);
void GPIO_ENCODER(void);
void GPIO_MOTOR(void);

#endif /* DMTB_GPIO_H_ */
```

C.12 DMTB_GPIO.c

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_GPIO.c
/**********************************/

#include "DMTB.h"

void GPIO_INIT(void)
{
    GPIO_LED();
    GPIO_PWM();
    GPIO_ENCODER();
    GPIO_MOTOR();
    GPIO_ADC();

}

void GPIO_ADC()
{
    GpioCtrlRegs.AIOMUX1.bit.AIO2 = 2;     // Configure AIO2 for A2 (analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO4 = 2;     // Configure AIO4 for A4 (analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO6 = 2;     // Configure AIO6 for A6 (analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO10 = 2;    // Configure AIO10 for B2 (analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO12 = 2;    // Configure AIO12 for B4 (analog input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO14 = 2;    // Configure AIO14 for B6 (analog input) operation
}

void GPIO_LED(void)
{
    // Set-up GPIO39 and GPIO40 as output, set low

    // gpio33 pULSE
    GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 0;     // GPIO39 is GPIO
    GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0;      // Enable pull-up on GPIO39
    GpioCtrlRegs.GPBDIR.bit.GPIO33 = 1;      // GPIO39 is output
    GpioDataRegs.GPBCLEAR.bit.GPIO33 = 1;    // INIT as OFF

    //LED uC GREEN
    GpioCtrlRegs.GPBMUX1.bit.GPIO39 = 0;     // GPIO39 is GPIO
    GpioCtrlRegs.GPBPUD.bit.GPIO39 = 0;      // Enable pull-up on GPIO39
```

```
45      GpioCtrlRegs.GPBDIR.bit.GPIO39 = 1;      // GPIO39 is output
46      GpioDataRegs.GPBCLEAR.bit.GPIO39 = 1;    // INIT as OFF
47
48      //LED uC RED
49      GpioCtrlRegs.GPBMUX1.bit.GPIO40 = 0;     // GPIO40 is GPIO
50      GpioCtrlRegs.GPBPUD.bit.GPIO40 = 0;      // Enable pull-up on GPIO40
51      GpioCtrlRegs.GPBDIR.bit.GPIO40 = 1;      // GPIO40 is output
52      GpioDataRegs.GPBCLEAR.bit.GPIO40 = 1;    // INIT as OFF
53
54      //LED 1
55      GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;      // GPIO7 is GPIO
56      GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0;       // Enable pull-up on GPIO40
57      GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;       // GPIO7 is output
58      GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;     // INIT as OFF
59      //GpioDataRegs.GPASET.bit.GPIO7 = 1;
60
61      //LED 2
62      GpioCtrlRegs.GPBMUX1.bit.GPIO44 = 0;     // GPIO44 is GPIO
63      GpioCtrlRegs.GPBPUD.bit.GPIO44 = 0;      // Enable pull-up on GPIO40
64      GpioCtrlRegs.GPBDIR.bit.GPIO44 = 1;      // GPIO44 is output
65      GpioDataRegs.GPBCLEAR.bit.GPIO44 = 1;    // INIT as OFF
66
67      //LED 3
68      GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0;     // GPIO16 is GPIO
69      GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0;      // Enable pull-up on GPIO40
70      GpioCtrlRegs.GPADIR.bit.GPIO16 = 1;      // GPIO16 is output
71      GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;    // INIT as OFF
72
73      //LED 4
74      GpioCtrlRegs.GPBMUX2.bit.GPIO52 = 0;     // GPIO52 is GPIO
75      GpioCtrlRegs.GPBPUD.bit.GPIO52 = 0;      // Enable pull-up on GPIO40
76      GpioCtrlRegs.GPBDIR.bit.GPIO52 = 1;      // GPIO52 is output
77      GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;    // INIT as OFF
78
79      //LED 5
80      GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0;     // GPIO17 is GPIO
81      GpioCtrlRegs.GPAPUD.bit.GPIO17 = 0;      // Enable pull-up on GPIO40
82      GpioCtrlRegs.GPADIR.bit.GPIO17 = 1;      // GPIO17 is output
83      GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;    // INIT as OFF
84
85      //LED 6
86      GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0;     // GPIO18 is GPIO
87      GpioCtrlRegs.GPAPUD.bit.GPIO18 = 0;      // Enable pull-up on GPIO40
88      GpioCtrlRegs.GPADIR.bit.GPIO18 = 1;      // GPIO18 is output
89      GpioDataRegs.GPACLEAR.bit.GPIO18 = 1;    // INIT as OFF
90
91  }
92
93  void GPIO_PWM(void)
94  {
95      // Set-up GPIO0 and GPIO2 as PWM
96      GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1;    // Disable pull-up on GPIO0
97      GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1;    // Disable pull-up on GPIO2
98      GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // GPIO0 = PWM1A
```

```
99      GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // GPIO2 = PWM2A
100
101     // Set-up GPIO4 and GPIO6 as PWM
102     GpioCtrlRegs.GPAPUD.bit.GPIO4 = 1;    // Disable pull-up on GPIO4
103     GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1;    // Disable pull-up on GPIO6
104     GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;   // GPIO4 = PWM3A
105     GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;   // GPIO6 = PWM4A
106 }
107
108 void GPIO_ENCODER(void)
109 {
110     // Set-up GPIO 10 and 11 as encoder SSI clocks
111     GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;     // GPIO8
112     GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;      // Disable pull-up on GPIO8
113     GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;      // GPIO8 = output
114     GpioDataRegs.GPASET.bit.GPIO11 = 1;    // Init as High
115
116     GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0;     // GPIO10
117     GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;      // Disable pull-up on GPIO10
118     GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;      // GPIO10 = output
119     GpioDataRegs.GPASET.bit.GPIO10 = 1;    // Init as high
120
121     // Set up GPIO 9 and 8 as encoder SSI data inputs
122     GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 0;     // GPIO9
123     GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;      // Disable pull-up on GPIO9
124     GpioCtrlRegs.GPADIR.bit.GPIO9 = 0;      // GPIO9 = input
125
126     GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0;     // GPIO11
127     GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;      // Disable pull-up on GPIO11
128     GpioCtrlRegs.GPADIR.bit.GPIO8 = 0;      // GPIO11 = input
129 }
130
131 void GPIO_MOTOR(void)
132 {
133     // Set-up GPIO 42, 43, and 44 as motor enables (outputs), set low
134
135     //Enable 1
136     GpioCtrlRegs.GPBMUX1.bit.GPIO42 = 0;     // GPIO42
137     GpioCtrlRegs.GPBPUD.bit.GPIO42 = 0;      // Enable pull-up on GPIO42
138     GpioCtrlRegs.GPBDIR.bit.GPIO42 = 1;      // GPIO42 = output
139     GpioDataRegs.GPBCLEAR.bit.GPIO42 = 1;    // Init as disabled
140
141     //Enable 2
142     GpioCtrlRegs.GPBMUX1.bit.GPIO43 = 0;     // GPIO43
143     GpioCtrlRegs.GPBPUD.bit.GPIO43 = 0;      // Enable pull-up on GPIO43
144     GpioCtrlRegs.GPBDIR.bit.GPIO43 = 1;      // GPIO43 = output
145     GpioDataRegs.GPBCLEAR.bit.GPIO43 = 1;    // Init as Disabled
146
147     //Enable 3
148     GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 0;     // GPIO25
149     GpioCtrlRegs.GPAPUD.bit.GPIO25 = 0;      // Enable pull-up on GPIO43
150     GpioCtrlRegs.GPADIR.bit.GPIO25 = 1;      // GPIO25 = output
151     GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;    // Init as disabled
152
```

```
153     //Enable 4
154     GpioCtrlRegs.GPAMUX2.bit.GPIO31 = 0;     // GPIO31
155     GpioCtrlRegs.GPAPUD.bit.GPIO31 = 0;      // Enable pull-up on GPIO43
156     GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;      // GPIO31 = output
157     GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;    // Init as disabled
158
159 }
```

C.13 DMTB_LED.h

```
1  /**********************************/
2  /* Jacob Sobering
3  /* Master's Thesis Research
4  /* Georgia Institute of Technology
5  /* 2017-2018
6  /*
7  /* DMTB_LED.h
8  /**********************************/
9
10 #ifndef DMTB_LED_H_
11 #define DMTB_LED_H_
12
13 void LED_ALL_ON(void);
14 void LED_ALL_OFF(void);
15 void LED_ON(Uint8 lednum);
16 void LED_OFF(Uint8 lednum);
17
18 #endif /* DMTB_LED_H_ */
```

C.14 DMTB_LED.c

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_LED.c
/**********************************/

#include "DMTB.h"

void LED_ALL_ON(void)
{
    GpioDataRegs.GPBSET.bit.GPIO39 = 1;    //GREEN
    GpioDataRegs.GPBSET.bit.GPIO40 = 1;    //RED
    GpioDataRegs.GPASET.bit.GPIO7 = 1;     //LED1
    GpioDataRegs.GPBSET.bit.GPIO44 = 1;    //LED2
    GpioDataRegs.GPASET.bit.GPIO16 = 1;    //LED3
    GpioDataRegs.GPBSET.bit.GPIO52 = 1;    //LED4
    GpioDataRegs.GPASET.bit.GPIO17 = 1;    //LED5
    GpioDataRegs.GPASET.bit.GPIO18 = 1;    //LED6

}

void LED_ALL_OFF(void)
{
    GpioDataRegs.GPBCLEAR.bit.GPIO39 = 1;    //GREEN (7)
    GpioDataRegs.GPBCLEAR.bit.GPIO40 = 1;    //RED (8)
    GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;     //LED1
    GpioDataRegs.GPBCLEAR.bit.GPIO44 = 1;    //LED2
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;    //LED3
    GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;    //LED4
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;    //LED5
    GpioDataRegs.GPACLEAR.bit.GPIO18 = 1;    //LED6
}

void LED_ON(Uint8 lednum)
{
    switch (lednum)
    {
    case 1:
        GpioDataRegs.GPASET.bit.GPIO7 = 1;
        break;
    case 2:
        GpioDataRegs.GPBSET.bit.GPIO44 = 1;
        break;
    case 3:
        GpioDataRegs.GPASET.bit.GPIO16 = 1;
        break;
    case 4:
```

```
51          GpioDataRegs.GPBSET.bit.GPIO52 = 1;
52          break;
53      case 5:
54          GpioDataRegs.GPASET.bit.GPIO17 = 1;
55          break;
56      case 6:
57          GpioDataRegs.GPASET.bit.GPIO18 = 1;
58          break;
59      case 7:
60          GpioDataRegs.GPBSET.bit.GPIO39 = 1;
61          break;
62      case 8:
63          GpioDataRegs.GPBSET.bit.GPIO40 = 1;
64          break;
65      }
66  }
67
68  void LED_OFF(Uint8 lednum)
69  {
70      switch (lednum)
71      {
72      case 1:
73          GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;
74          break;
75      case 2:
76          GpioDataRegs.GPBCLEAR.bit.GPIO44 = 1;
77          break;
78      case 3:
79          GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
80          break;
81      case 4:
82          GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
83          break;
84      case 5:
85          GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
86          break;
87      case 6:
88          GpioDataRegs.GPACLEAR.bit.GPIO18 = 1;
89          break;
90      case 7:
91          GpioDataRegs.GPBCLEAR.bit.GPIO39 = 1;
92          break;
93      case 8:
94          GpioDataRegs.GPBCLEAR.bit.GPIO40 = 1;
95          break;
96      }
97  }
```

C.15 DMTB_MOTOR.h

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_Motor.h
/**********************************/

#ifndef DMTB_MOTOR_H_
#define DMTB_MOTOR_H_

void MOTOR_ENABLE_CONTROLLER(void);
void MOTOR_DISABLE_CONTROLLER(void);
void MOTOR_ENABLE_LOAD(void);
void MOTOR_DISABLE_LOAD(void);
void MOTOR_ENABLE(void);
void MOTOR_DISABLE(void);

#endif /* DMTB_MOTOR_H_ */
```

C.16 DMTB_MOTOR.c

```
/***********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_Motor.c
/***********************************/

#include "DMTB.h"

void MOTOR_ENABLE_CONTROLLER(void)
{
    GpioDataRegs.GPBSET.bit.GPIO42 = 1;
    GpioDataRegs.GPBSET.bit.GPIO43 = 1;
}

void MOTOR_DISABLE_CONTROLLER(void)
{
    GpioDataRegs.GPBCLEAR.bit.GPIO42 = 1;
    GpioDataRegs.GPBCLEAR.bit.GPIO43 = 1;
}

void MOTOR_ENABLE_LOAD(void)
{
    GpioDataRegs.GPASET.bit.GPIO25 = 1;
    GpioDataRegs.GPASET.bit.GPIO31 = 1;
}

void MOTOR_DISABLE_LOAD(void)
{
    GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
}

void MOTOR_ENABLE(void)
{
    MOTOR_ENABLE_CONTROLLER();
    MOTOR_ENABLE_LOAD();
}

void MOTOR_DISABLE(void)
{
    MOTOR_DISABLE_CONTROLLER();
    MOTOR_DISABLE_LOAD();
}
```

C.17 DMTB_PWM.h

```
1  /**********************************/
2  /* Jacob Sobering
3  /* Master's Thesis Research
4  /* Georgia Institute of Technology
5  /* 2017-2018
6  /*
7  /* DMTB_PWM.h
8  /**********************************/
9
10 #ifndef DMTB_PWM_H_
11 #define DMTB_PWM_H_
12
13 void PWM_INIT(void);
14 void PWM_TimerSetup(void);
15 void PWM_INIT1(void);
16 void PWM_INIT2(void);
17
18 #endif /* DMTB_PWM_H_ */
```

C.18 DMTB_PWM.c

```
/**********************************/
/* Jacob Sobering
/* Master's Thesis Research
/* Georgia Institute of Technology
/* 2017-2018
/*
/* DMTB_PWM.c
/**********************************/

#include "DMTB.h"

void PWM_INIT(void)
{
    SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;             //Enable ePWM1 clock
    SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;             //Enable ePWM2 clock
    SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;             //Enable ePWM3 clock
    SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;             //Enable ePWM4 clock

    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;              //Disable sync (re-enable later)

    PWM_INIT1();
    PWM_INIT2();

    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
}

void PWM_INIT1(void)
{
    /**********EPWMA*********/
    EPwm1Regs.TBPRD = FULLPWM; //1500

    EPwm1Regs.CMPA.half.CMPA = HALFPWM;      //750                 //50% duty cycle is 0V, 100% is 12V, 0% is -12V)

    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0;                  //Timer Divide by 1 (30kHz)
    EPwm1Regs.TBCTL.bit.CLKDIV = 0;                     //Timer Divide by 1

    EPwm1Regs.AQCTLA.bit.CAU  = 1;                      //AQ_CLEAR
    EPwm1Regs.AQCTLA.bit.CAD = 2;                       //AQ_SET

    EPwm1Regs.ETSEL.bit.SOCAEN = 1;
    EPwm1Regs.ETSEL.bit.SOCASEL = 5;   //1 - Zero, 2 -  PRD, 5 - CMPA when Decrementing
```

```
42      EPwm1Regs.ETPS.bit.SOCAPRD  = 3;
43
44      EPwm1Regs.TBCTL.bit.CTRMODE = 2;                    //Timer Count up
            -Down mode
45
46      /***********EPWMB*********/
47      EPwm2Regs.TBPRD = FULLPWM;
48
49      EPwm2Regs.CMPA.half.CMPA = HALFPWM;                    //50% duty
            cycle is 0V, 100% is 12V, 0% is -12V)
50
51      EPwm2Regs.TBCTL.bit.HSPCLKDIV = 0;                  //Timer Divide
            by 1 (30kHz)
52      EPwm2Regs.TBCTL.bit.CLKDIV = 0;                     //Timer Divide
            by 1
53
54      EPwm2Regs.AQCTLA.bit.CAU  = 2;
55      EPwm2Regs.AQCTLA.bit.CAD = 1;
56      EPwm2Regs.TBCTL.bit.CTRMODE = 2;                    //Timer Count up
            -Down mode
57  }
58
59  void PWM_INIT2(void)
60  {
61      /***********EPWMC*********/
62      EPwm3Regs.TBPRD = FULLPWM;
63
64      EPwm3Regs.CMPA.half.CMPA = HALFPWM;                    //50% duty
            cycle is 0V, 100% is 12V, 0% is -12V)
65
66      EPwm3Regs.TBCTL.bit.HSPCLKDIV = 0;                  //Timer Divide
            by 1 (30kHz)
67      EPwm3Regs.TBCTL.bit.CLKDIV = 0;                     //Timer Divide
            by 1
68
69      EPwm3Regs.AQCTLA.bit.CAU  = 1;
70      EPwm3Regs.AQCTLA.bit.CAD = 2;
71      EPwm3Regs.TBCTL.bit.CTRMODE = 2;                    //Timer Count up
            -Down mode
72
73      /***********EPWMD*********/
74      EPwm4Regs.TBPRD = FULLPWM;
75
76      EPwm4Regs.CMPA.half.CMPA = HALFPWM;                    //50% duty
            cycle is 0V, 100% is 12V, 0% is -12V)
77
78      EPwm4Regs.TBCTL.bit.HSPCLKDIV = 0;                  //Timer Divide
            by 1 (30kHz)
79      EPwm4Regs.TBCTL.bit.CLKDIV = 0;                     //Timer Divide
            by 1
80
81      EPwm4Regs.AQCTLA.bit.CAU  = 2;
82      EPwm4Regs.AQCTLA.bit.CAD = 1;
```

```
83      EPwm4Regs.TBCTL.bit.CTRMODE = 2;                    //Timer Count up
            -Down mode
84  }
```

REFERENCES

[1] C. W. Alcorn, M. A. Croom, M. S. Francis, and H. Ross, "The X-31 Aircraft: Advances In Aircraft Agility and Performance," *Progress in Aerospace Sciences*, pp. 377–413, 1996.

[2] *The F-1 Engine Powered Apollo Into History, Blazes Path for Space Launch System Advanced Propulsion*, `https://www.nasa.gov/topics/history/features/f1_engine.html`, Accessed: 11-13-2017.

[3] B. L. Berrier and J. G. Taylor, "Internal Performance of Two Nozzles Utilizing Gimbal Concepts for Thrust Vectoring," *NASA Technical Paper 2991*, 1990.

[4] M. R. Schaefermeyer, "Aerodynamic Thrust Vectoring For Attitude Control Of A Vertically Thrusting Jet Engine," *AIAA*, 2011.

[5] B. L. Deere K. A. abd Berrie, J. D. Flamm, and S. K. Johnson, "Computational Study of Fluidic Thrust Vectoring Using Separation Control in a Nozzle," *AIAA*, 2003.

[6] T. Dungan, *A-4/V-2 Makeup - Tech Data and Markings*, `http://www.v2rocket.com/start/makeup/design.html`, Accessed: 11-15-2017.

[7] *V-2 rocket, howpublished = https://en.wikipedia.org/wiki/v-2_rocket*, Accessed: 11-17-2017.

[8] *Rockwell-MBB X-31, howpublished = https://en.wikipedia.org/wiki/rockwell-mbb_x-31*, Accessed: 11-17-2017.

[9] M. A. Al-Alaoui, "Novel Approach to Designing Digital Differentiators," 1992.

[10] "DCX16S GB KL 12V Datasheet," *Maxon Motor*, 2016.

[11] P. Brokaw, "An IC Amplifier User's Guide to Decoupling, Grounding, and Making Things Go Right For a Change," *Application Note AN-202*,